PA0

# GitHub

- GitHub is a cloud-based platform where you can store, share, and work together with others to write code.
- Storing your code in a "repository" on GitHub allows you to:
  - Showcase or share your work.
  - Track and manage changes to your code over time.
  - Let others review your code, and make suggestions to improve it.
  - Collaborate on a shared project, without worrying that your changes will impact the work of your collaborators before you're ready to integrate them.

Collaborative working, one of GitHub’s fundamental features, is made possible by the open-source software, Git, upon which GitHub is built.

# Git

- Git is a version control system that intelligently tracks changes in files. Git is particularly useful when you and a group of people are all making changes to the same files at the same time.
- Typically, to do this in a Git-based workflow, you would:
    - Create a branch off from the main copy of files that you (and your collaborators) are working on.
    - Make edits to the files independently and safely on your own personal branch.
    - Let Git intelligently merge your specific changes back into the main copy of files, so that your changes don't impact other people's updates.
    - Let Git keep track of your and other people's changes, so you all stay working on the most up-to-date version of the project.

# Git working with GitHub

- When you upload files to GitHub, you'll store them in a "Git repository." This means that when you make changes (or "commits") to your files in GitHub, Git will automatically start to track and manage your changes.
- People work on their files locally (on their own computer), then continually sync these local changes—and all the related Git data—with the central **"remote" repository** (where your code is stored) on GitHub.
- When you are ready to share your changes you use **push**, which means sending to the remote repository the new changes you made
- When you want to see the latest changes made by your collaborators from the remote repository, you use **pull**

# Commit vs Push

Commit:

- A commit records changes to your **local repository**. It creates a snapshot of the project's state at a specific point in time, along with a descriptive message explaining the changes.
- Commits are performed locally and do not immediately affect any remote repositories.
- You can create multiple commits locally before sharing them with others.

Push:

- A push transfers committed changes from your local repository to a **remote repository**. This makes your local commits available to other collaborators.
- Pushing effectively synchronizes your local history with the remote repository, allowing others to pull your changes and integrate them into their own local copies.

# Remote vs Local Repository

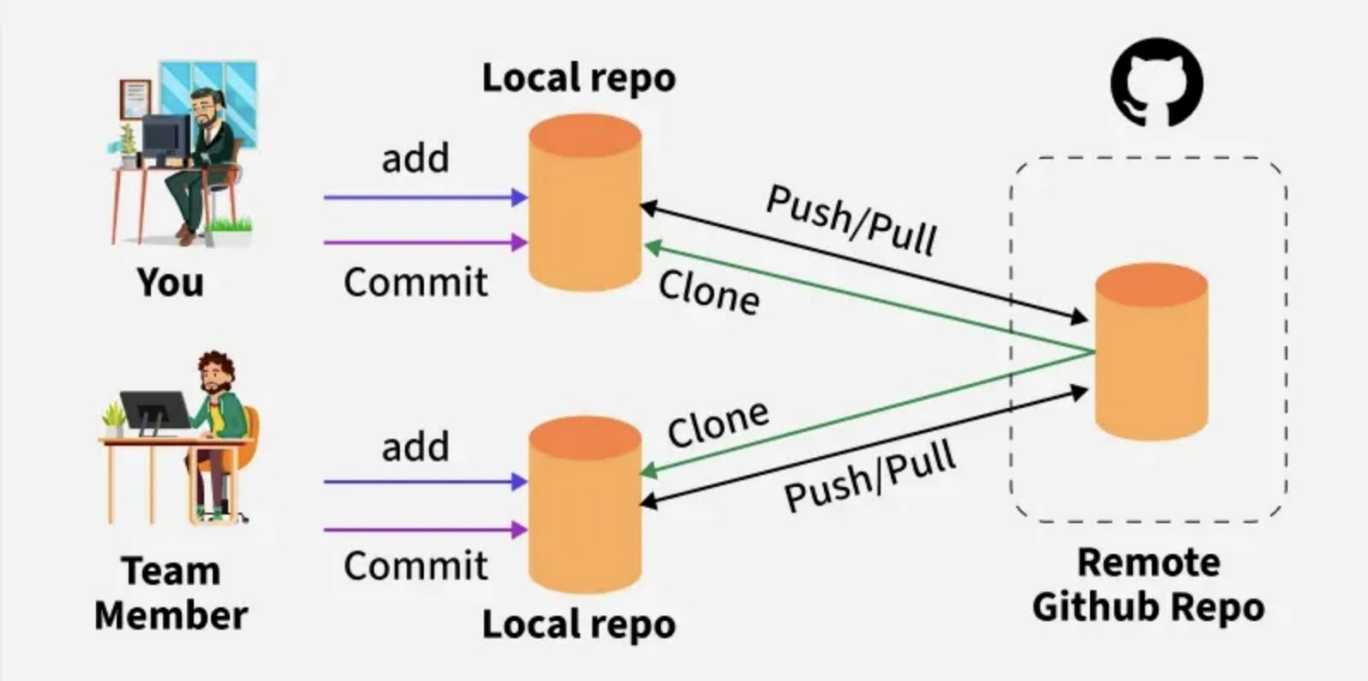
Local Repository:

- A local repository is a complete copy of a project's history stored on your local machine.
- Changes that are made in a local repository are not visible to other collaborators until you push to the remote repository

Remote Repository:

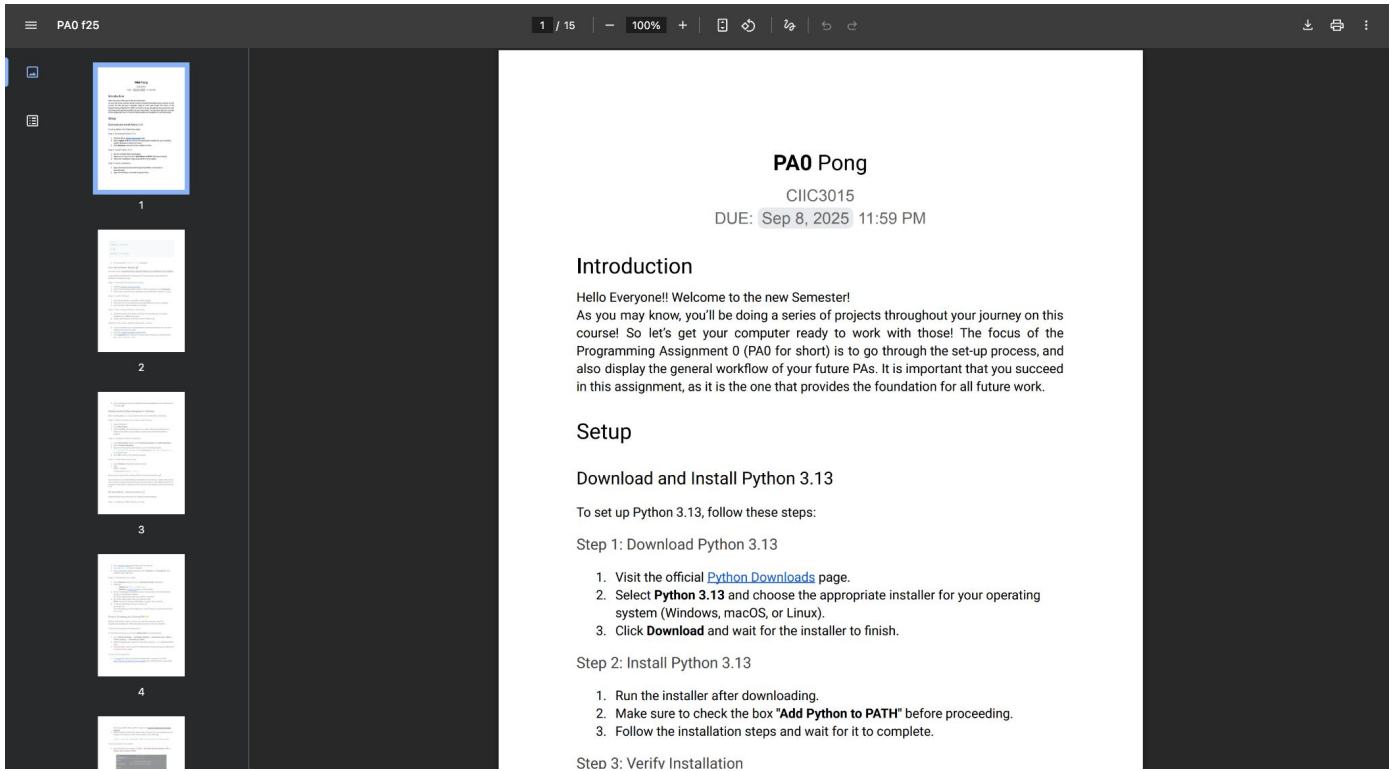
- A remote repository is a version of the project hosted on a server, typically on platforms like GitHub, GitLab, or Bitbucket.
- It serves as a central point for collaboration, allowing multiple developers to share and synchronize their work.

# Diagram




By: GeeksforGeeks

# Find the PA0 PDF in Moodle


PA0 f25
1 / 15
100%
1
2
3
4
PA0 Pong
CIIC3015
DUE: Sep 8, 2025 11:59 PM
Introduction
Hello Everyone!! Welcome to the new Semester!
As you may know, you'll be doing a series of projects throughout your journey on this course! So let's get your computer ready to work with those! The focus of the Programming Assignment 0 (PA0 for short) is to go through the set-up process, and also display the general workflow of your future PAs. It is important that you succeed in this assignment, as it is the one that provides the foundation for all future work.
Setup
Download and Install Python 3.13
To set up Python 3.13, follow these steps:
Step 1: Download Python 3.13
1. Visit the official Python Downloads page.
2. Select Python 3.13 and choose the appropriate installer for your operating system (Windows, macOS, or Linux).
3. Click Download and wait for the installer to finish.
Step 2: Install Python 3.13
1. Run the installer after downloading.
2. Make sure to check the box "Add Python to PATH" before proceeding.
3. Follow the installation steps and wait for it to complete.
Step 3: Verify Installation

# PA0 PDF Page 4

- First step is to accept the assignment following that link in Page 4

### Step 2: Configuring Git Locally

1. Open **Terminal** (macOS/Linux) or **Command Prompt** (Windows).
2. Install git:
   - **macOS:** Run `brew install git`
   - **Windows:** Download Git and run the installer.
- Run the following commands to set your Git username and email (replace with your actual GitHub details):
  git config --global user.name "your GitHub username"
3. git config --global user.email "your GitHub email"
   **NOTE:** You won't see any confirmation or output. This is normal.
4. To ensure everything is set up correctly, run:
   git config --list
   This will display your Git configuration—check that your username and email are correct.

## Project: Accepting and Cloning PA0 🤝

With Git and GitHub in place, it's time to accept and clone your very first programming assignment—PA0! Let's get your project onto your machine:

### Cloning the Assignment Repository

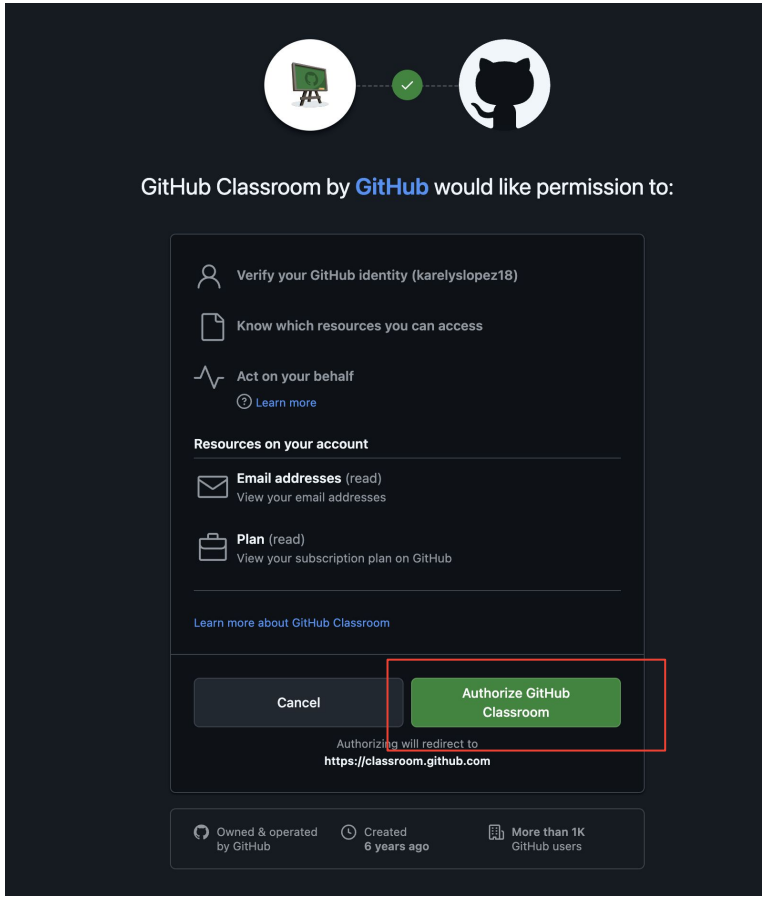
To clone the repository, you'll need a **GitHub token** for authentication.

1. Go to **GitHub's Settings → Developer Settings → Personal Access Tokens → Tokens (classic) → Generate new token**.
2. Select the appropriate scopes for your token (such as `repo`) and generate the token.
3. **Save the token**—you'll need it for authentication during cloning and GitHub will not show it to you again.

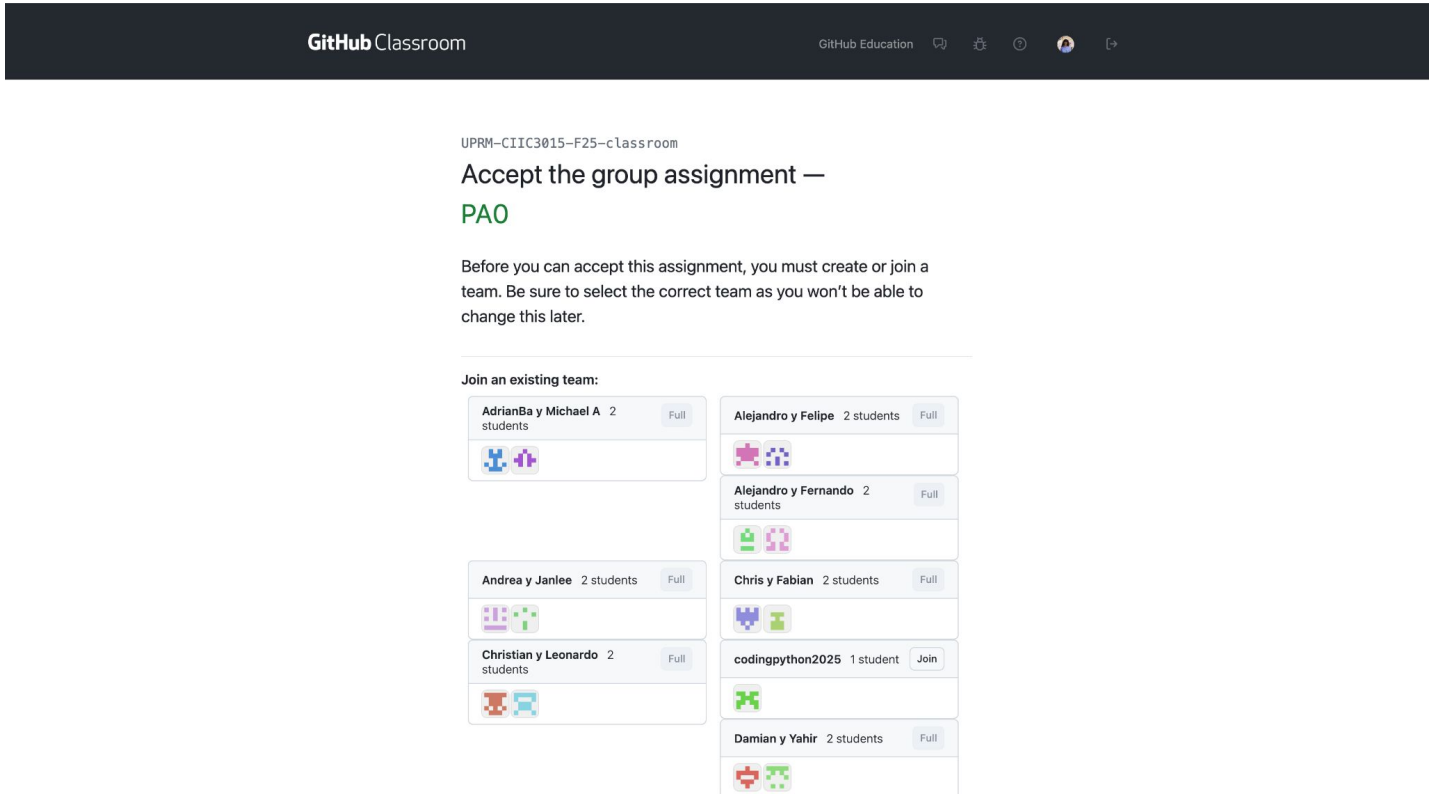
### Accept the Assignment:

1. Click here OR copy and paste the following link to accept or join PA0 https://classroom.github.com/a/selpBIG2. One of the partners must create

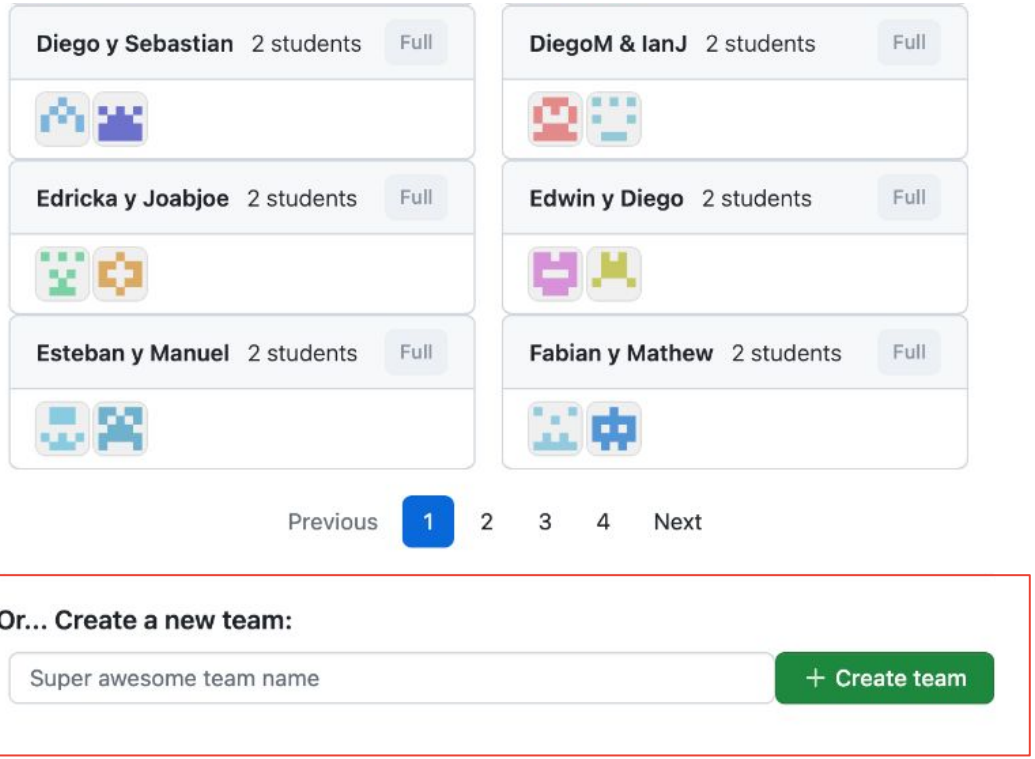
# Next step:


GitHub Classroom by GitHub would like permission to:
Verify your GitHub identity (karelyslopez18)
Know which resources you can access
Act on your behalf
Learn more
Resources on your account
Email addresses (read)
View your email addresses
Plan (read)
View your subscription plan on GitHub
Learn more about GitHub Classroom
Cancel
Authorize GitHub Classroom
Authorizing will redirect to
https://classroom.github.com
Owned & operated by GitHub
Created 6 years ago
More than 1K GitHub users

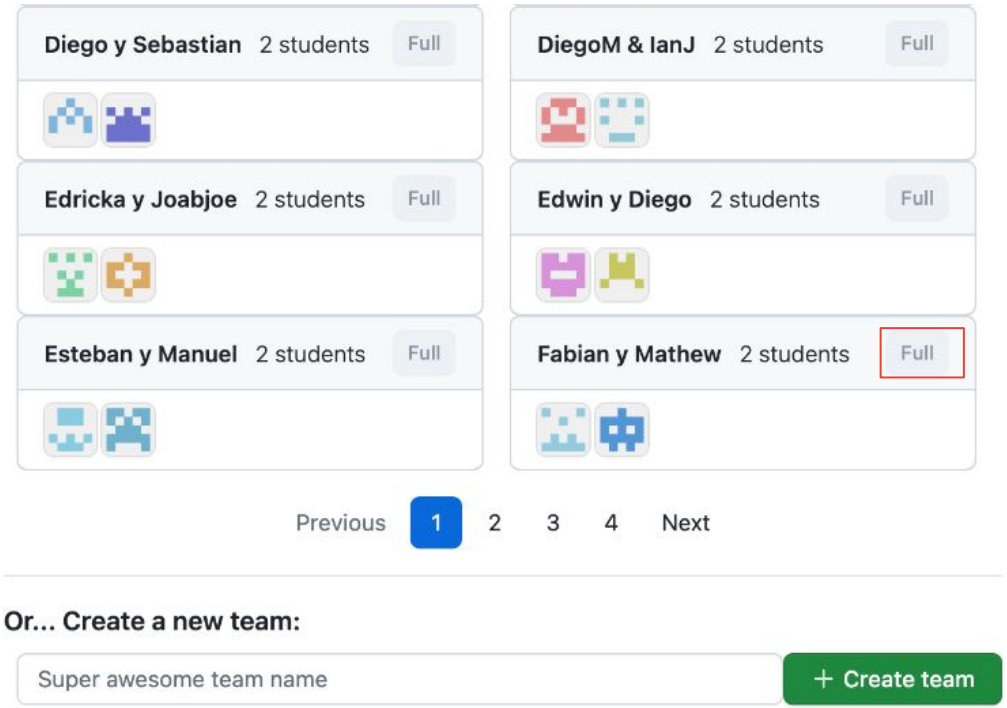
# Next step:


GitHub Classroom
GitHub Education
UPRM-CIIC3015-F25-classroom
Accept the group assignment —
PA0
Before you can accept this assignment, you must create or join a team. Be sure to select the correct team as you won't be able to change this later.
Join an existing team:
AdrianBa y Michael A 2 students
Full
Alejandro y Felipe 2 students
Full
Alejandro y Fernando 2 students
Full
Andrea y Janlee 2 students
Full
Chris y Fabian 2 students
Full
Christian y Leonardo 2 students
Full
codingpython2025 1 student
Join
Damian y Yahir 2 students
Full

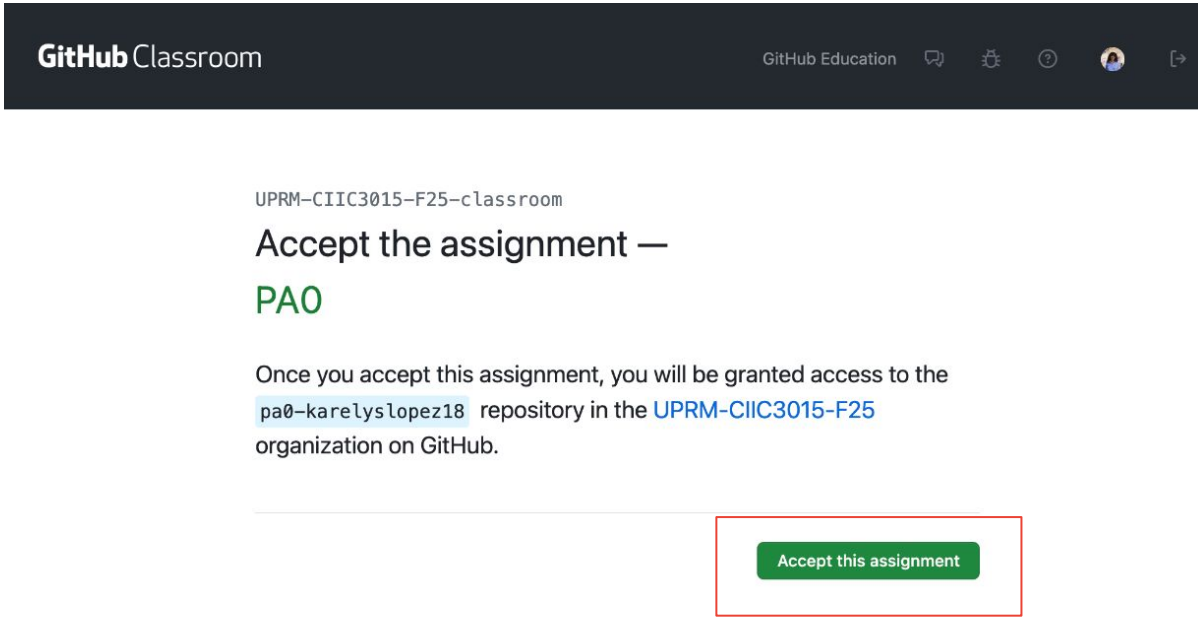
# If you are creating the team:


Diego y Sebastian 2 students Full
DiegoM & IanJ 2 students Full
Edricka y Joabjoe 2 students Full
Edwin y Diego 2 students Full
Esteban y Manuel 2 students Full
Fabian y Mathew 2 students Full
Previous 1 2 3 4 Next
Or... Create a new team:
Super awesome team name
+ Create team

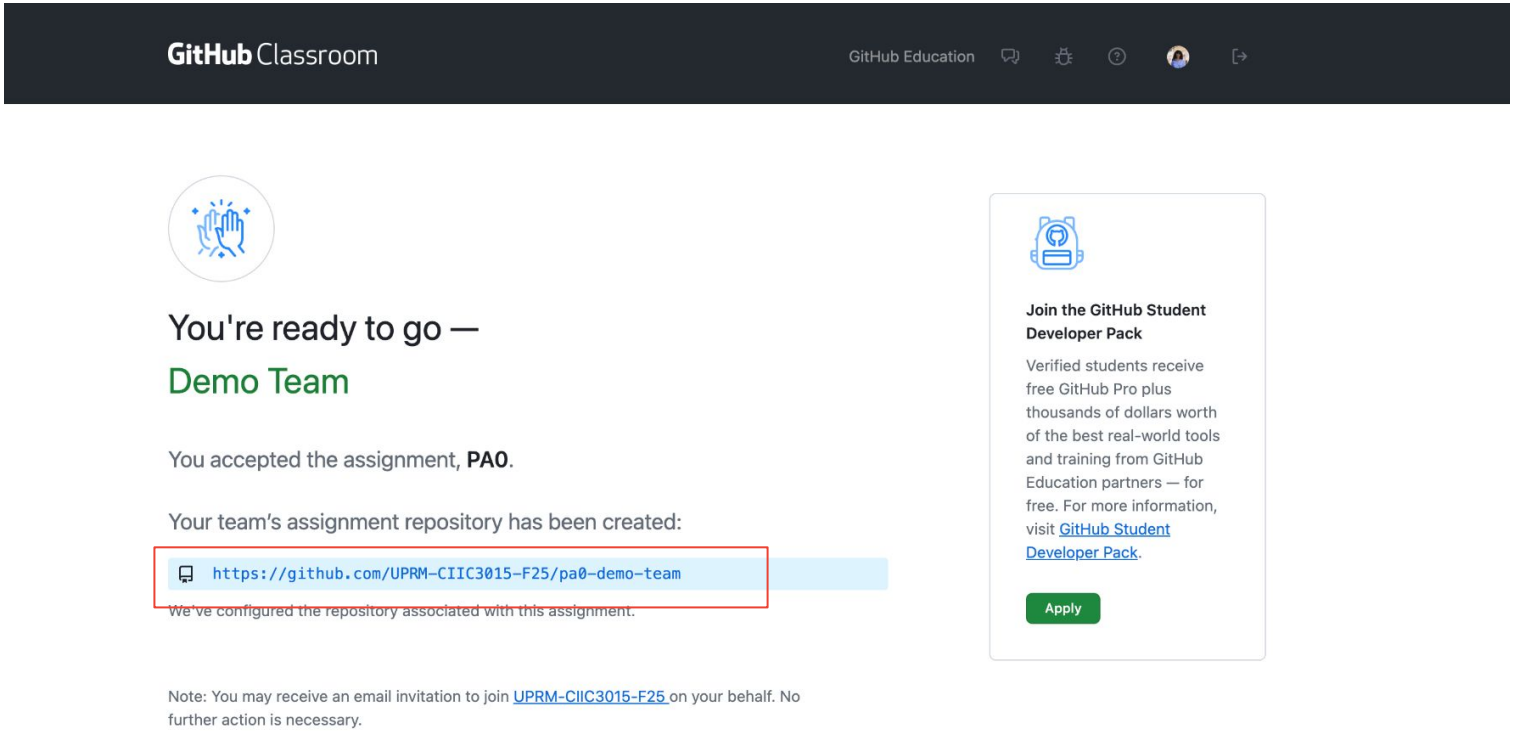
# If you are joining the team:


Diego y Sebastian 2 students Full
DiegoM & IanJ 2 students Full
Edricka y Joabjoe 2 students Full
Edwin y Diego 2 students Full
Esteban y Manuel 2 students Full
Fabian y Mathew 2 students Full
Previous 1 2 3 4 Next
Or... Create a new team:
Super awesome team name
+ Create team

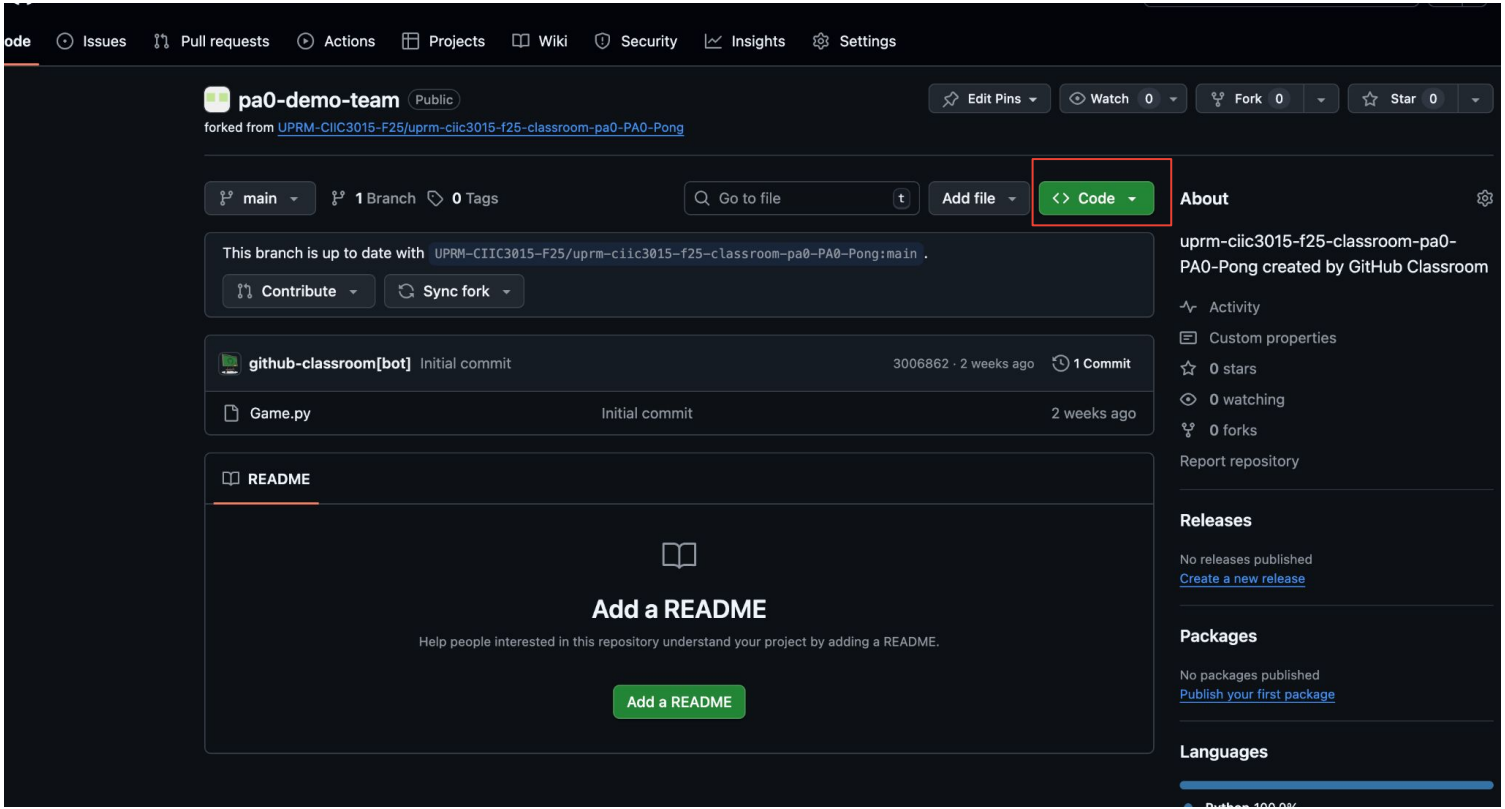
# Next Step:


GitHub Classroom
GitHub Education
UPRM-CIIC3015-F25-classroom
Accept the assignment —
PA0
Once you accept this assignment, you will be granted access to the pa0-karelyslopez18 repository in the UPRM-CIIC3015-F25 organization on GitHub.
Accept this assignment

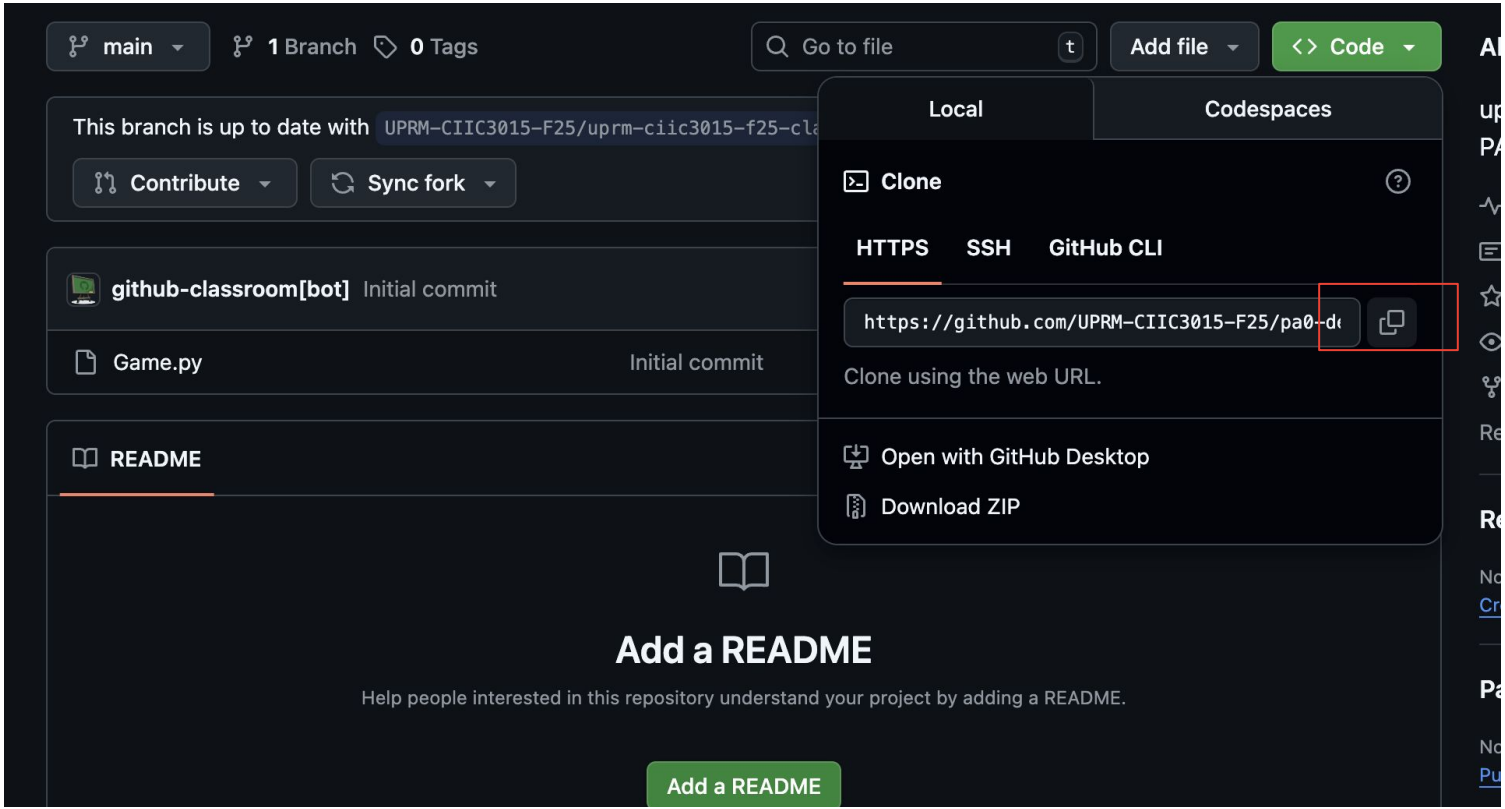
# Next Step:


GitHub Classroom
GitHub Education
You're ready to go —
Demo Team
You accepted the assignment, PA0.
Your team's assignment repository has been created:
https://github.com/UPRM-CIIC3015-F25/pa0-demo-team
We've configured the repository associated with this assignment.
Note: You may receive an email invitation to join UPRM-CIIC3015-F25 on your behalf. No further action is necessary.
Join the GitHub Student Developer Pack
Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. For more information, visit GitHub Student Developer Pack.
Apply

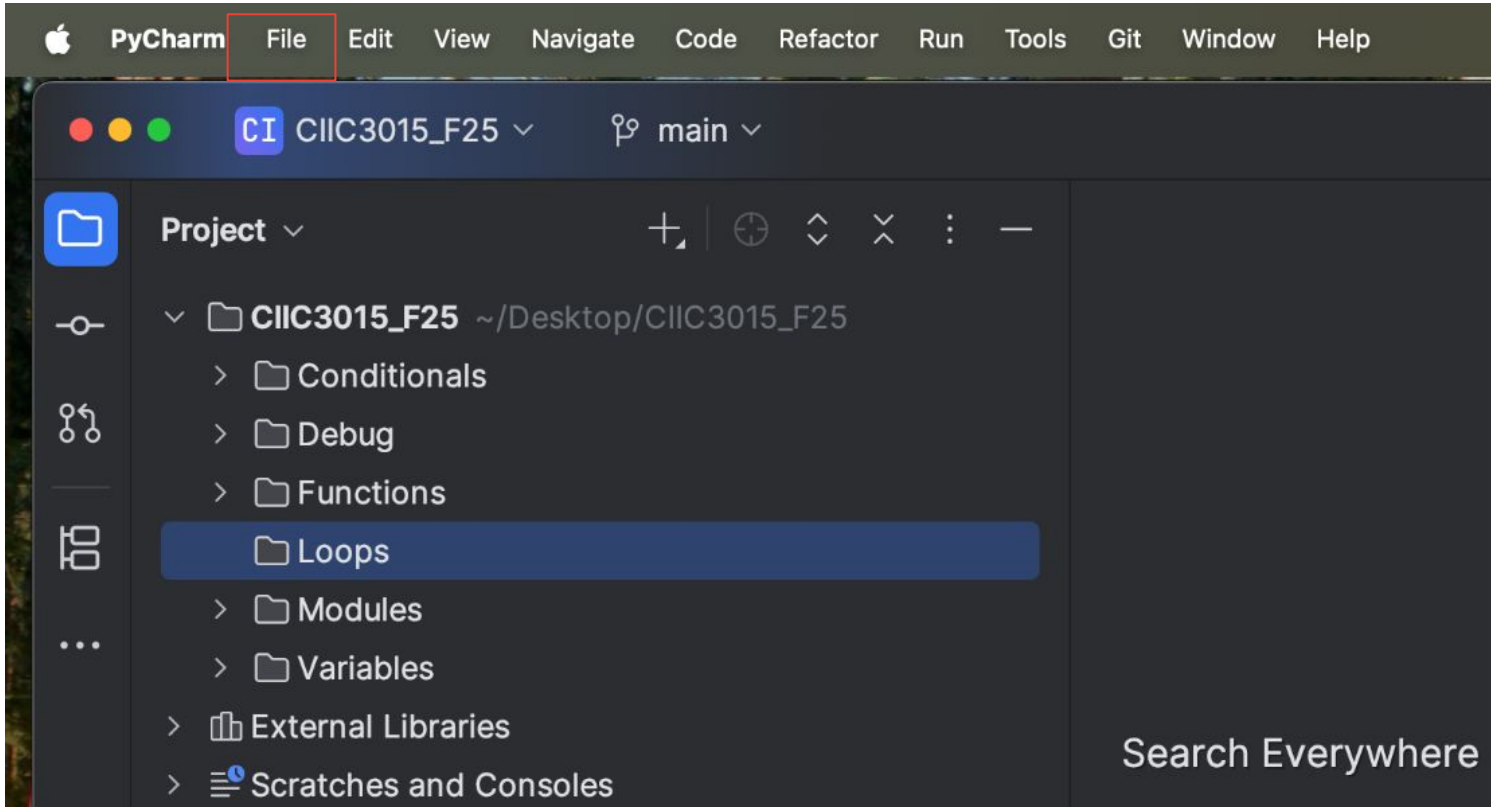
# Cloning the repo



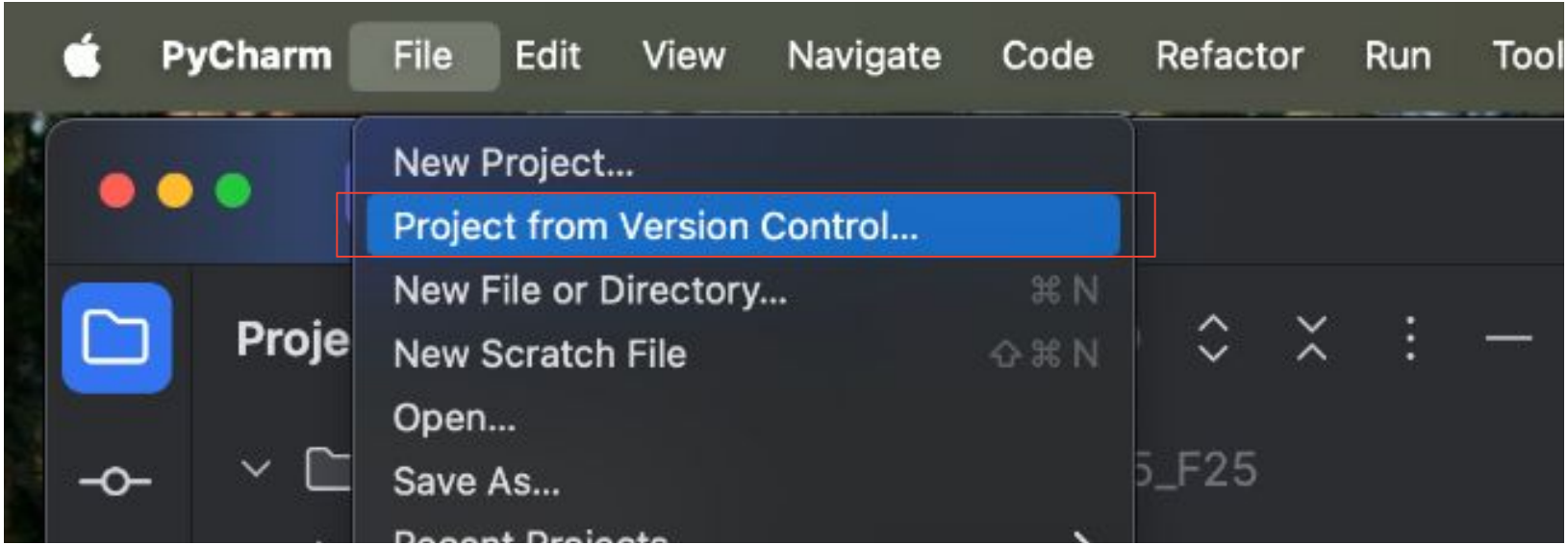
# Cloning the repo



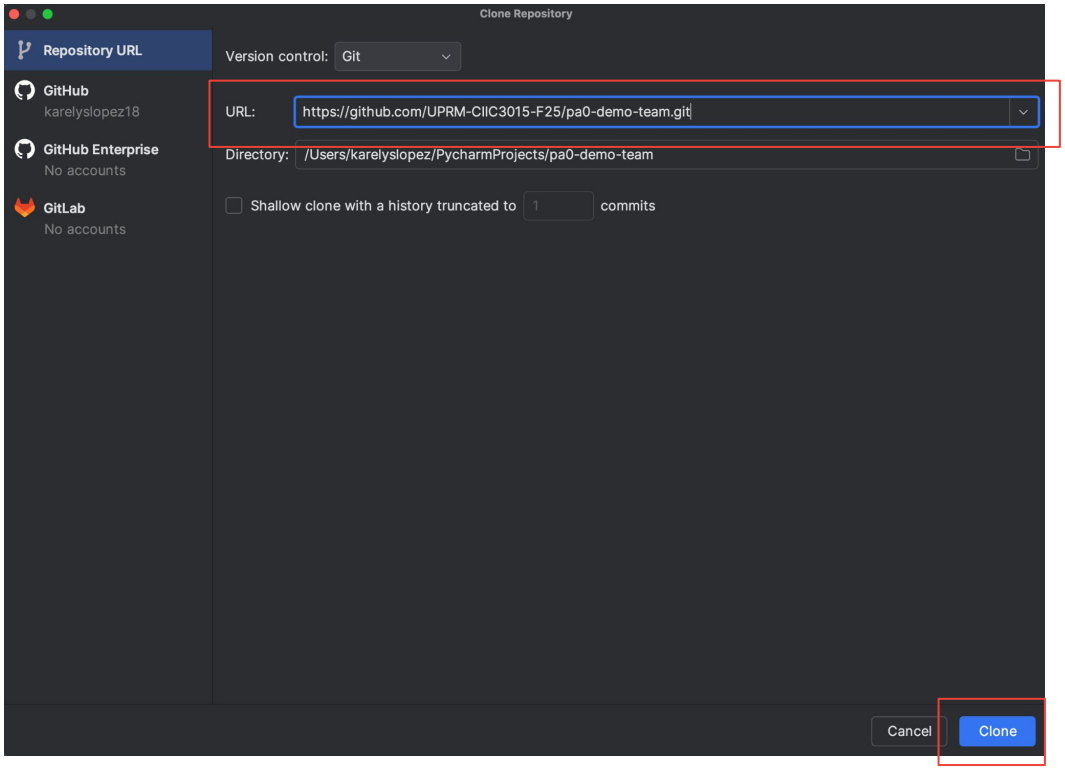
# Cloning the repo



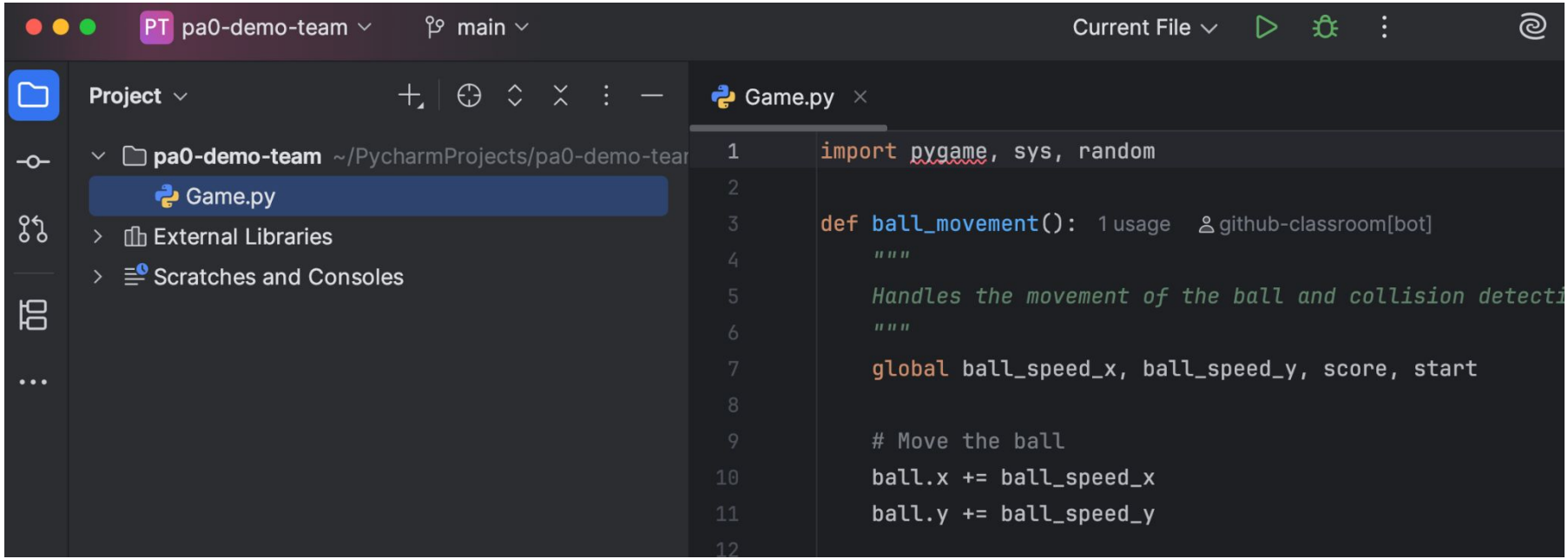
# Cloning the repo



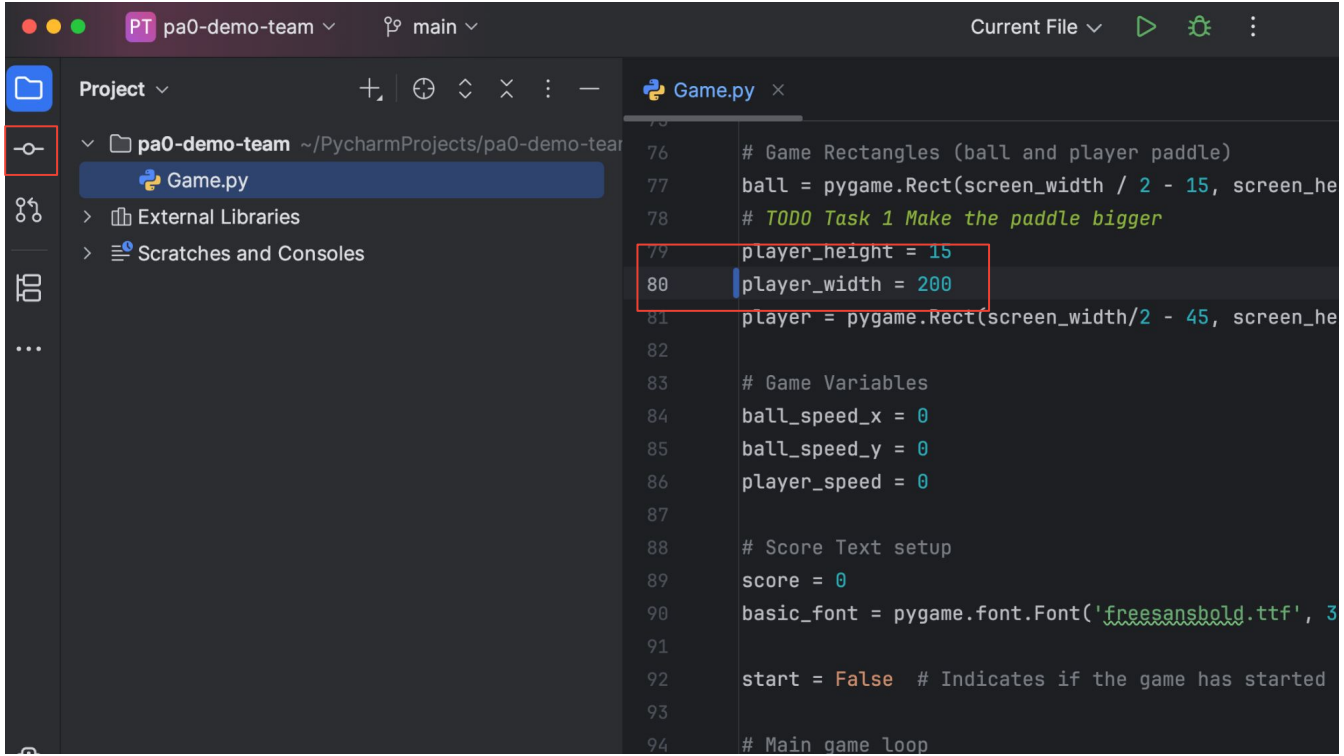
# Cloning the repo



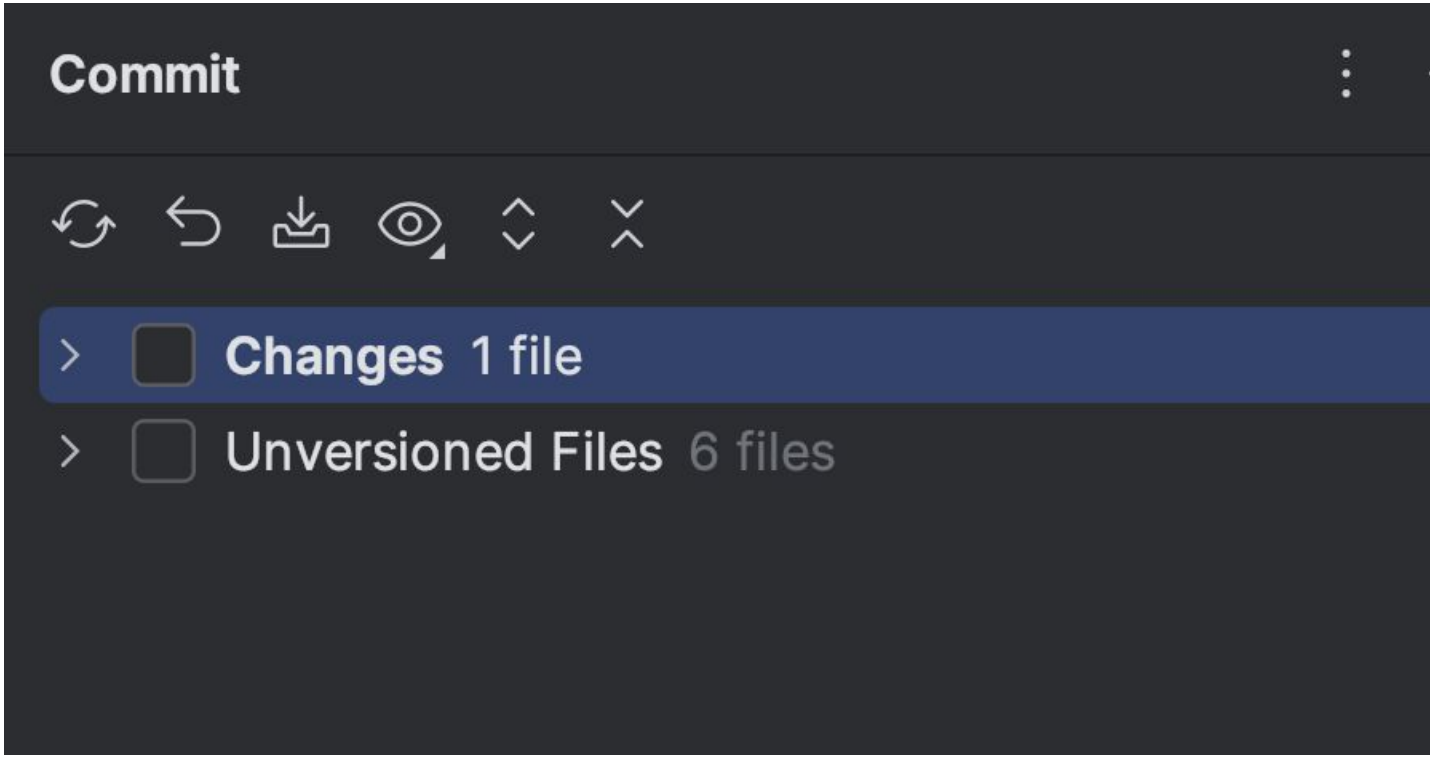
# Cloning the repo



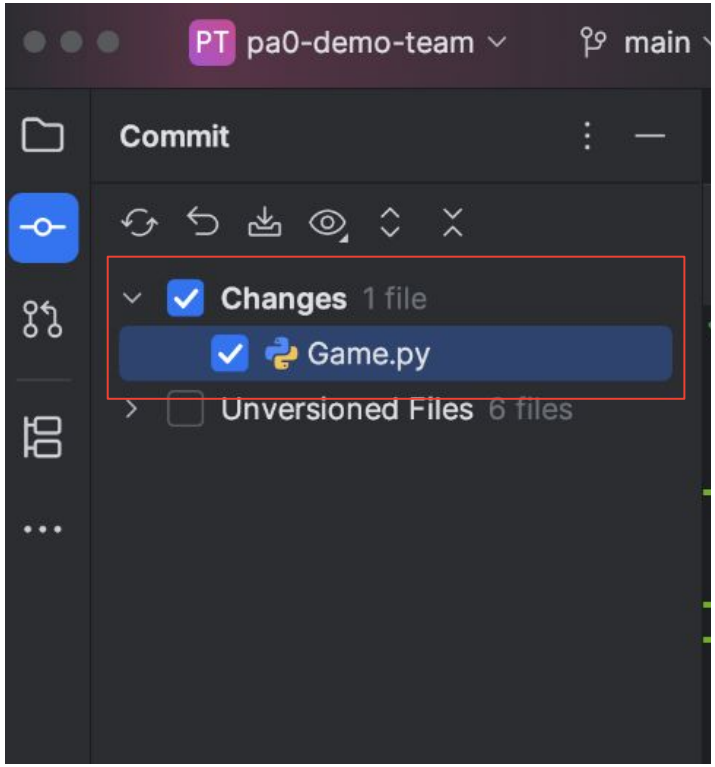
# Committing a Change



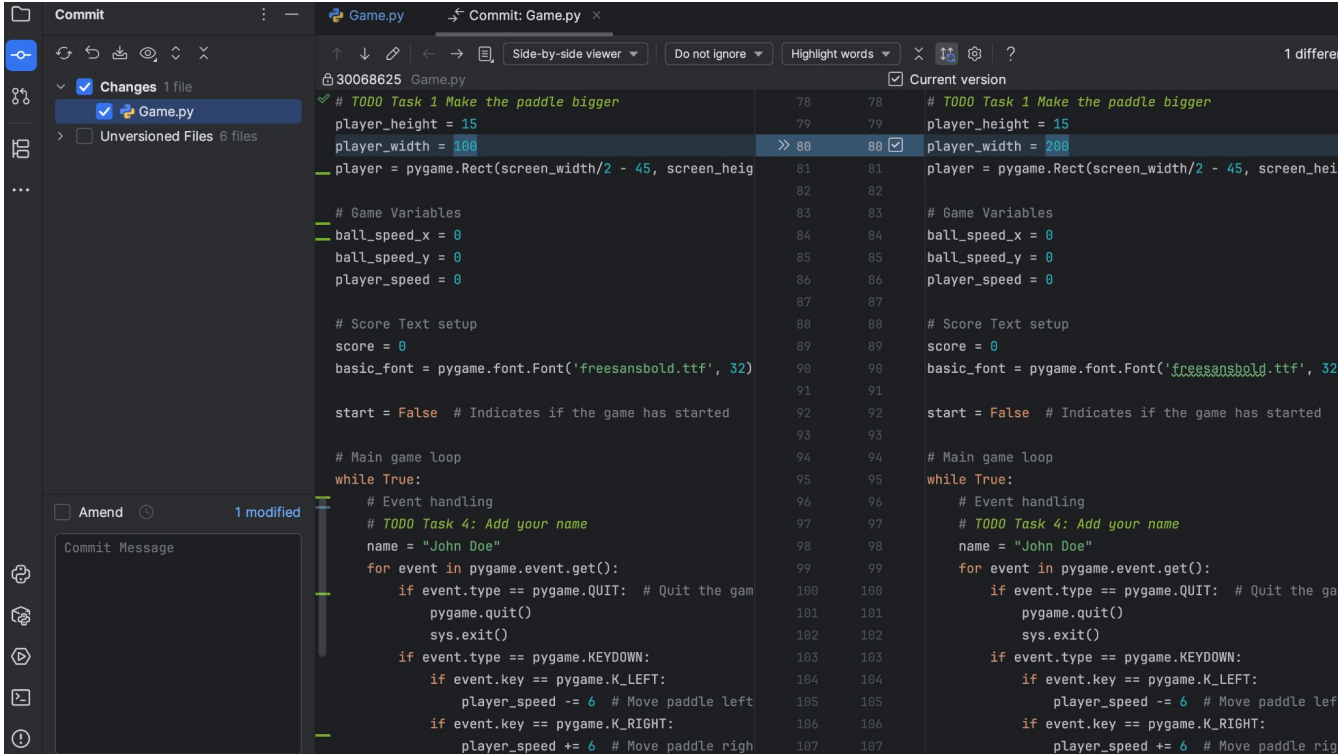
# Committing a Change



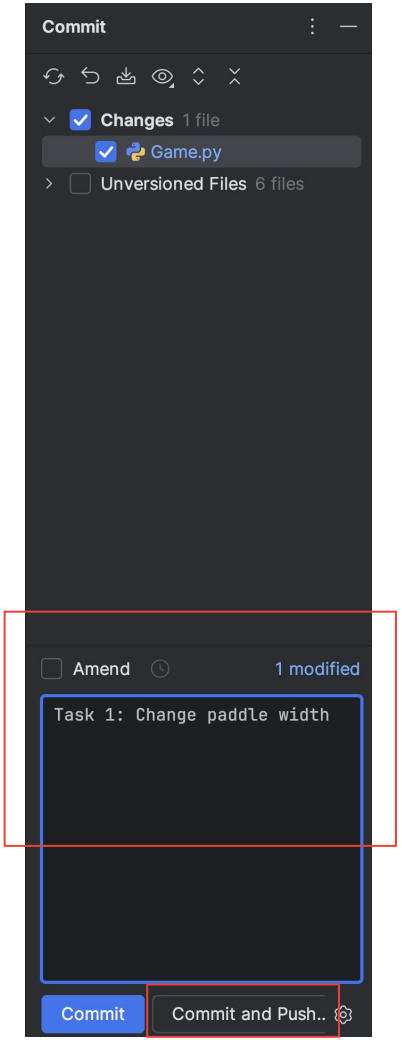
# Committing a Change



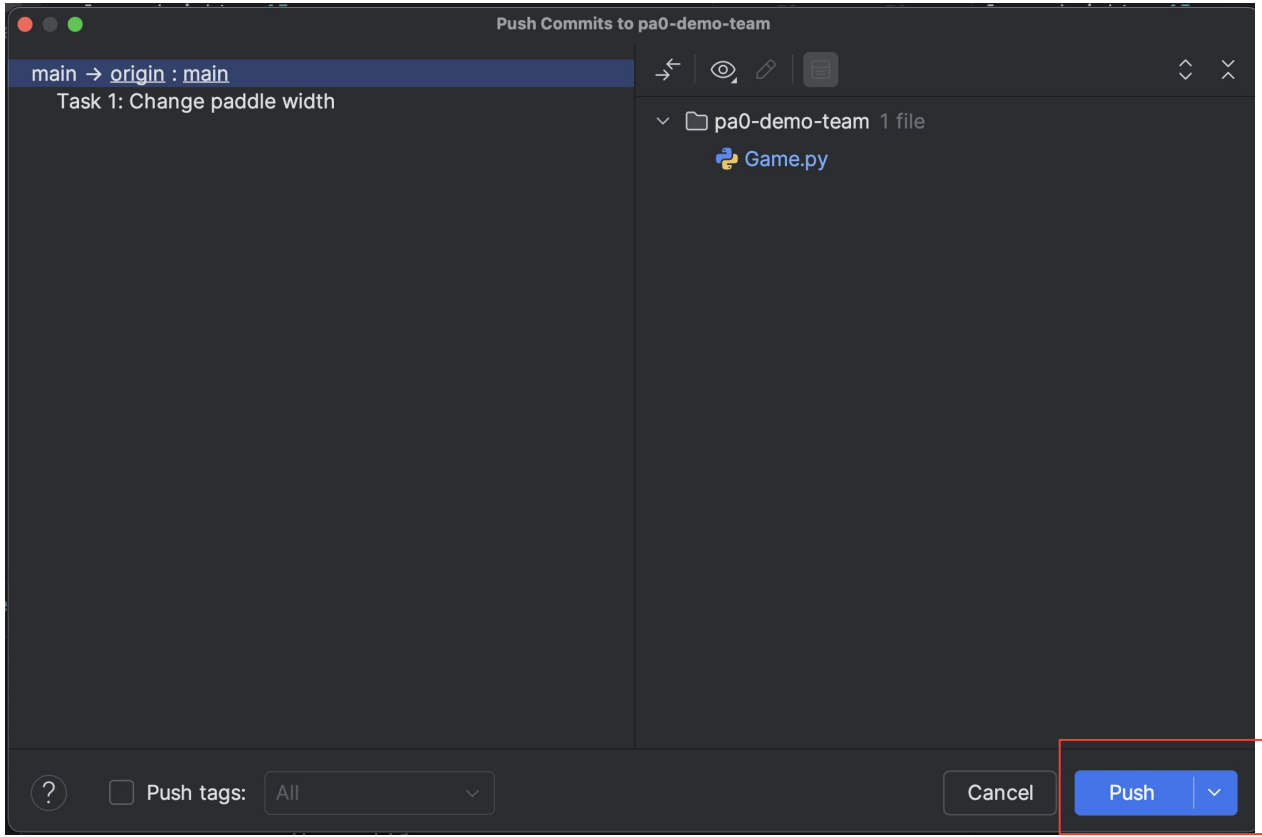
# Committing a Change (If you want to see the change you made)



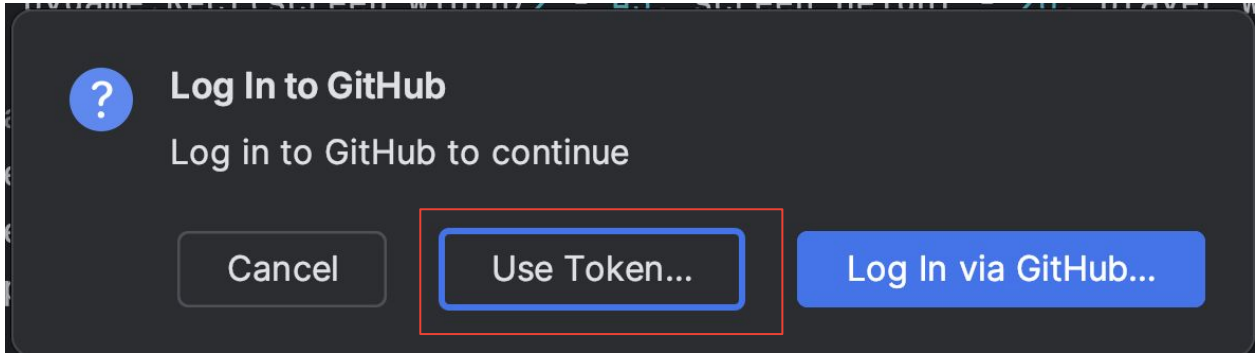
# Committing a Change



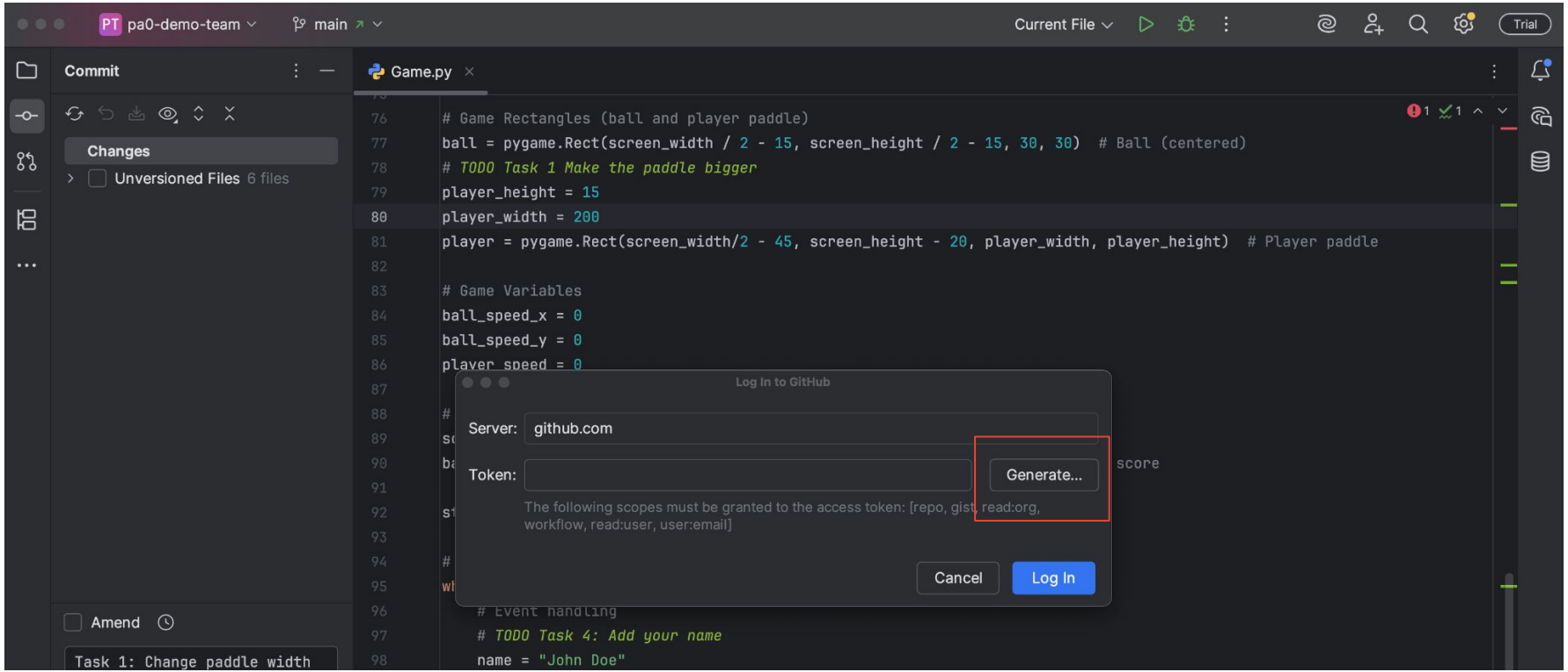
# Committing a Change



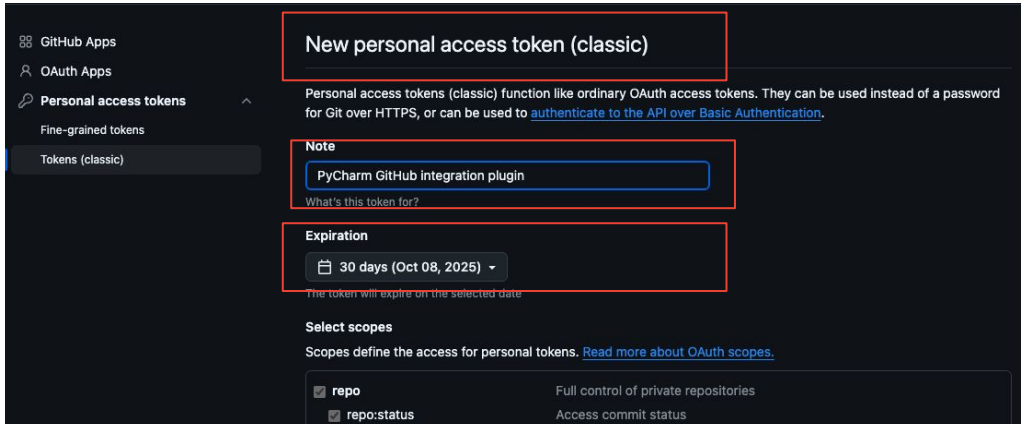
# Committing a Change



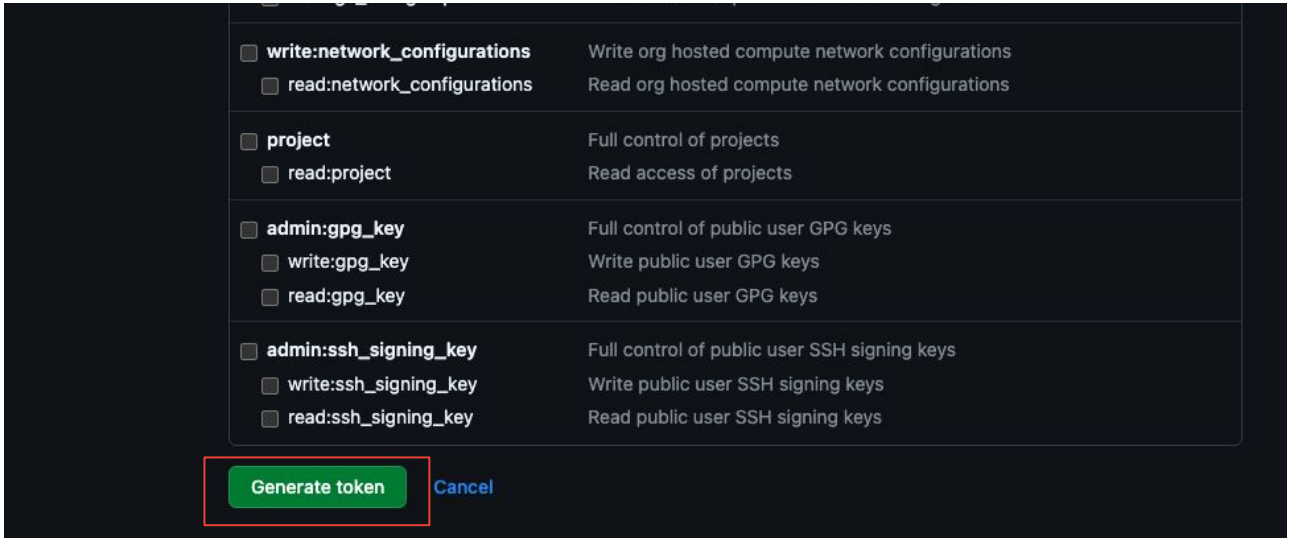
# Committing a Change


pa0-demo-team
main
Current File
Trial
Commit
Changes
Unversioned Files 6 files
Amend
Task 1: Change paddle width
Game.py
# Game Rectangles (ball and player paddle)
ball = pygame.Rect(screen_width / 2 - 15, screen_height / 2 - 15, 30, 30)  # Ball (centered)
# TODO Task 1 Make the paddle bigger
player_height = 15
player_width = 200
player = pygame.Rect(screen_width/2 - 45, screen_height - 20, player_width, player_height)  # Player paddle
# Game Variables
ball_speed_x = 0
ball_speed_y = 0
player_speed = 0
Log In to GitHub
Server: github.com
Token:
Generate...
The following scopes must be granted to the access token: [repo, gist, read:org, workflow, read:user, user:email]
Cancel
Log In
score
# Event handling
# TODO Task 4: Add your name
name = "John Doe"

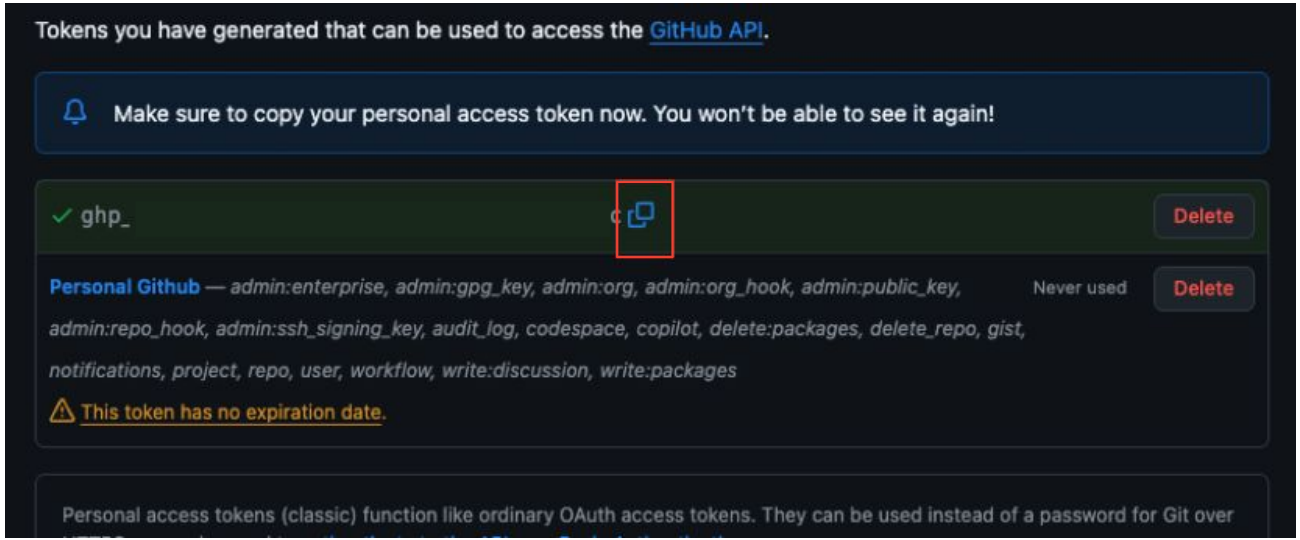
# Generate Token


GitHub Apps
OAuth Apps
Personal access tokens
Fine-grained tokens
Tokens (classic)
New personal access token (classic)
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.
Note
PyCharm GitHub integration plugin
What's this token for?
Expiration
30 days (Oct 08, 2025)
The token will expire on the selected date
Select scopes
Scopes define the access for personal tokens. Read more about OAuth scopes.
repo
Full control of private repositories
repo:status
Access commit status

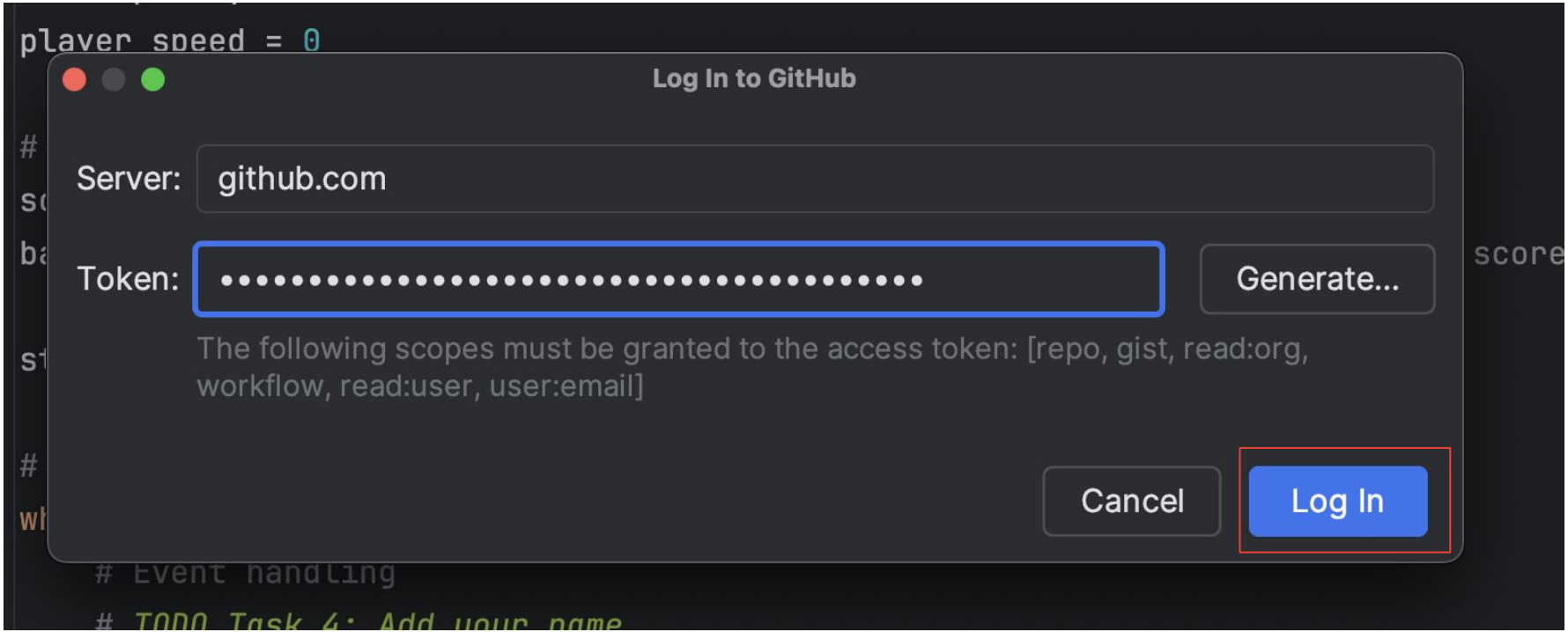
# Generate Token



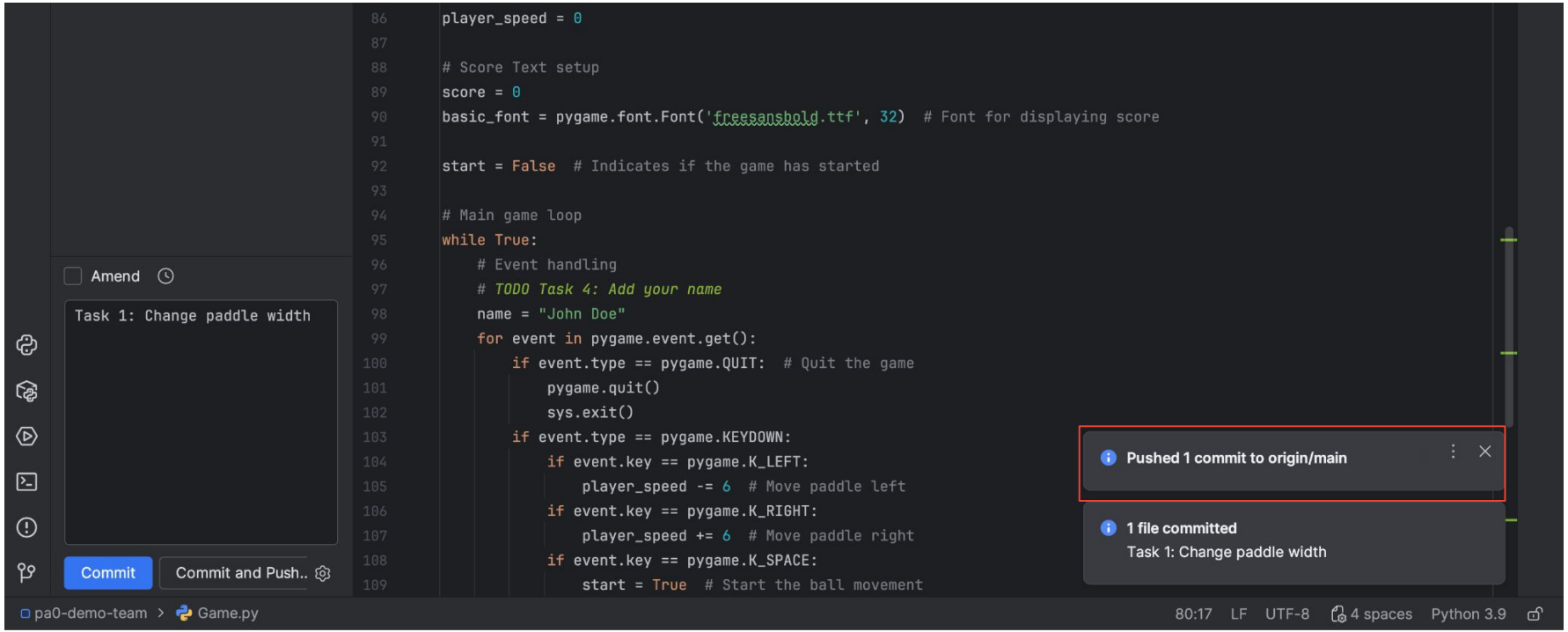
# Generate Token


Tokens you have generated that can be used to access the GitHub API.
Make sure to copy your personal access token now. You won't be able to see it again!
ghp_
Delete
Personal Github — admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages
Never used
Delete
This token has no expiration date.
Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over

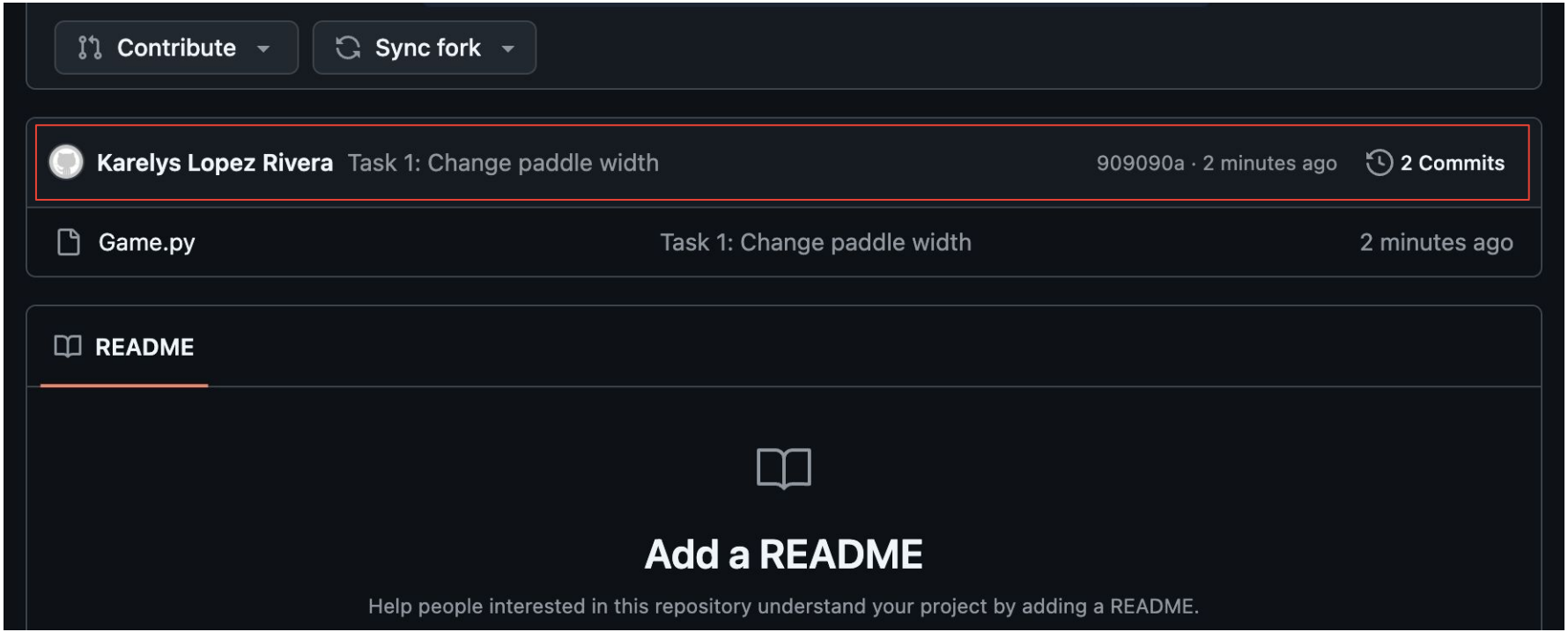
# Committing a Change



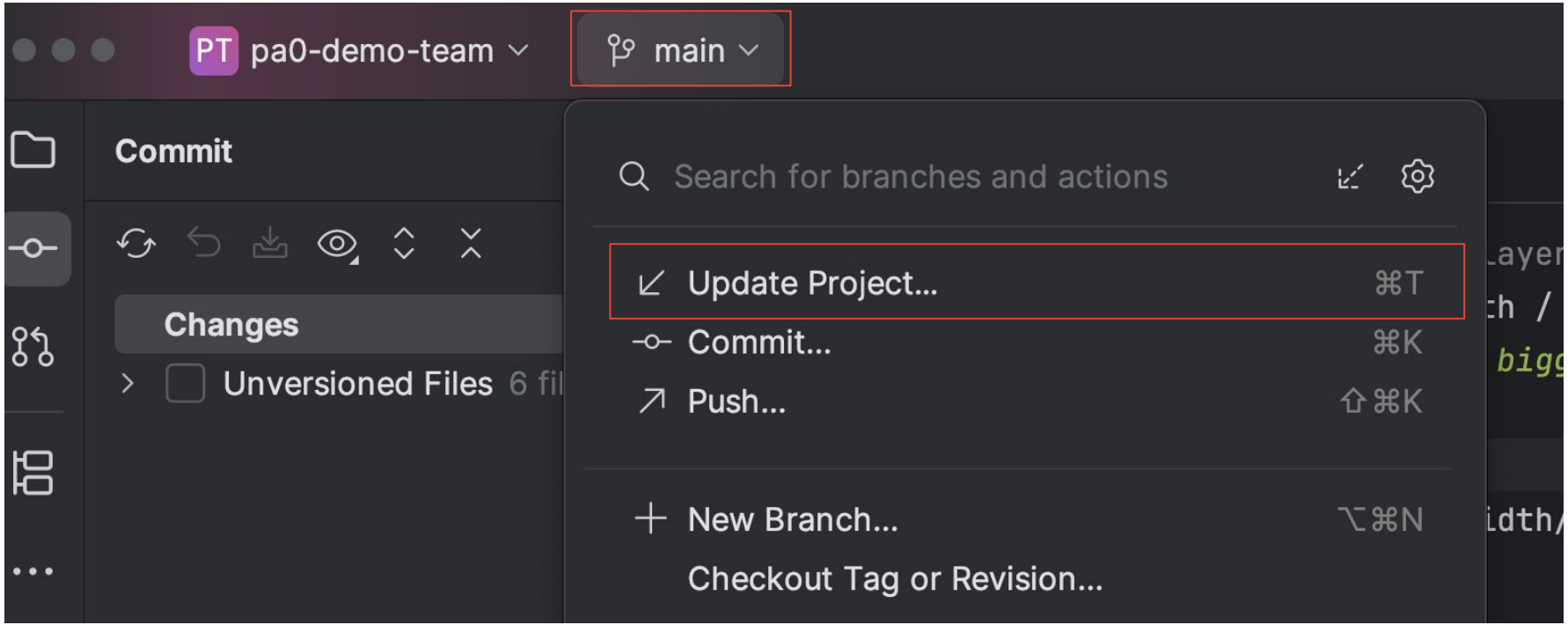
# Committing a Change



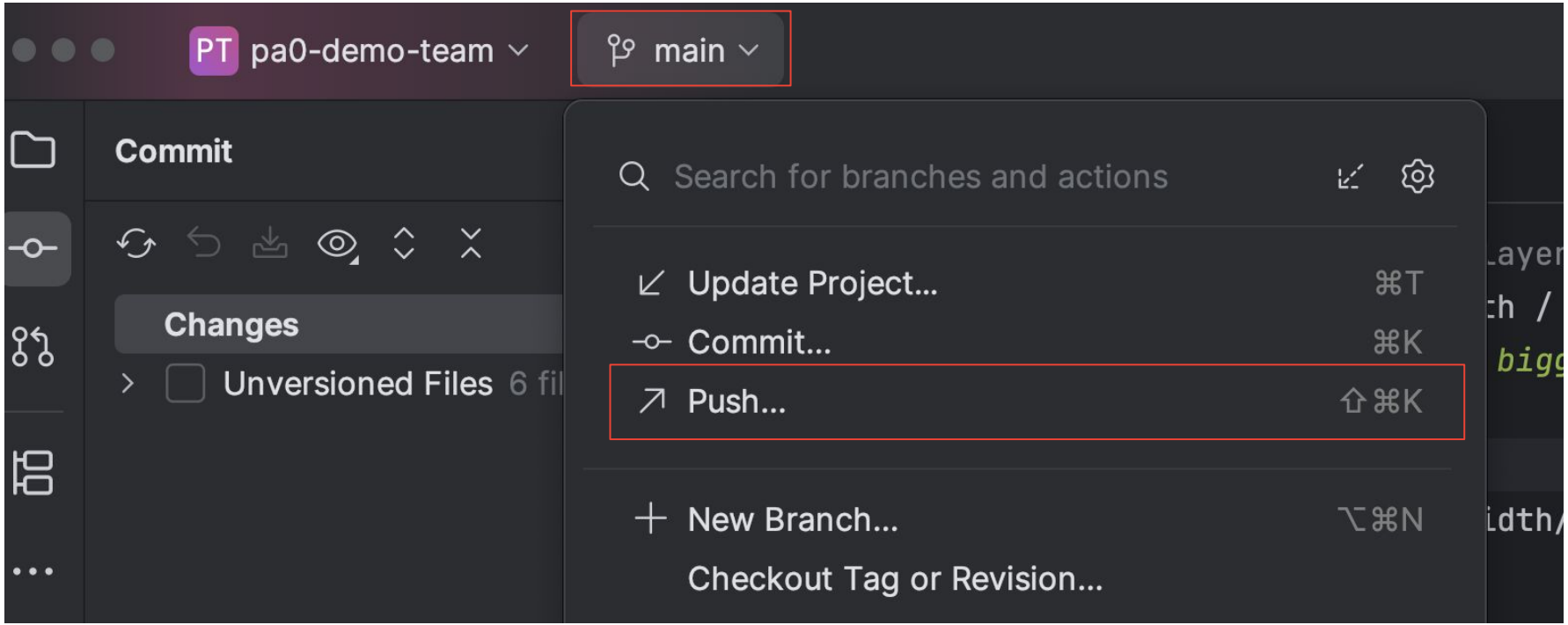
# Committing a Change


Contribute
Sync fork
Karelys Lopez Rivera Task 1: Change paddle width
909090a · 2 minutes ago
2 Commits
Game.py
Task 1: Change paddle width
2 minutes ago
README
Add a README
Help people interested in this repository understand your project by adding a README.

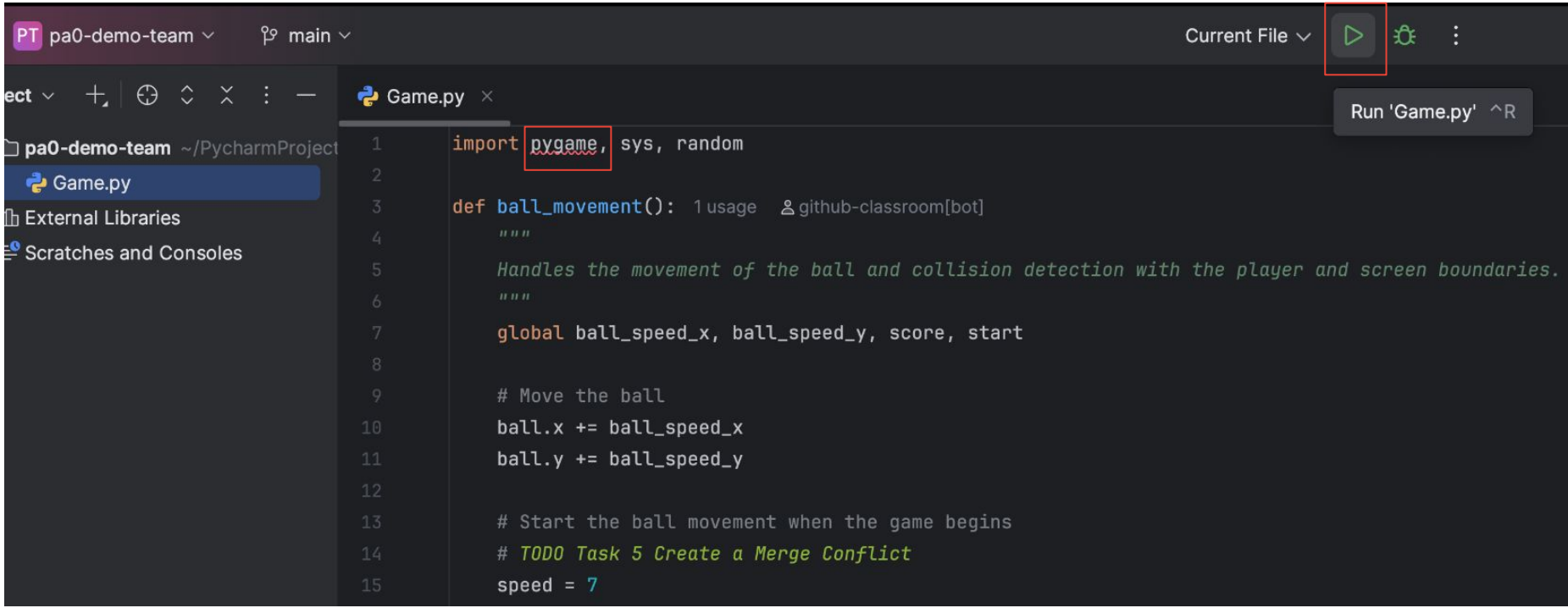
# Updating Project (Seeing teammate changes)



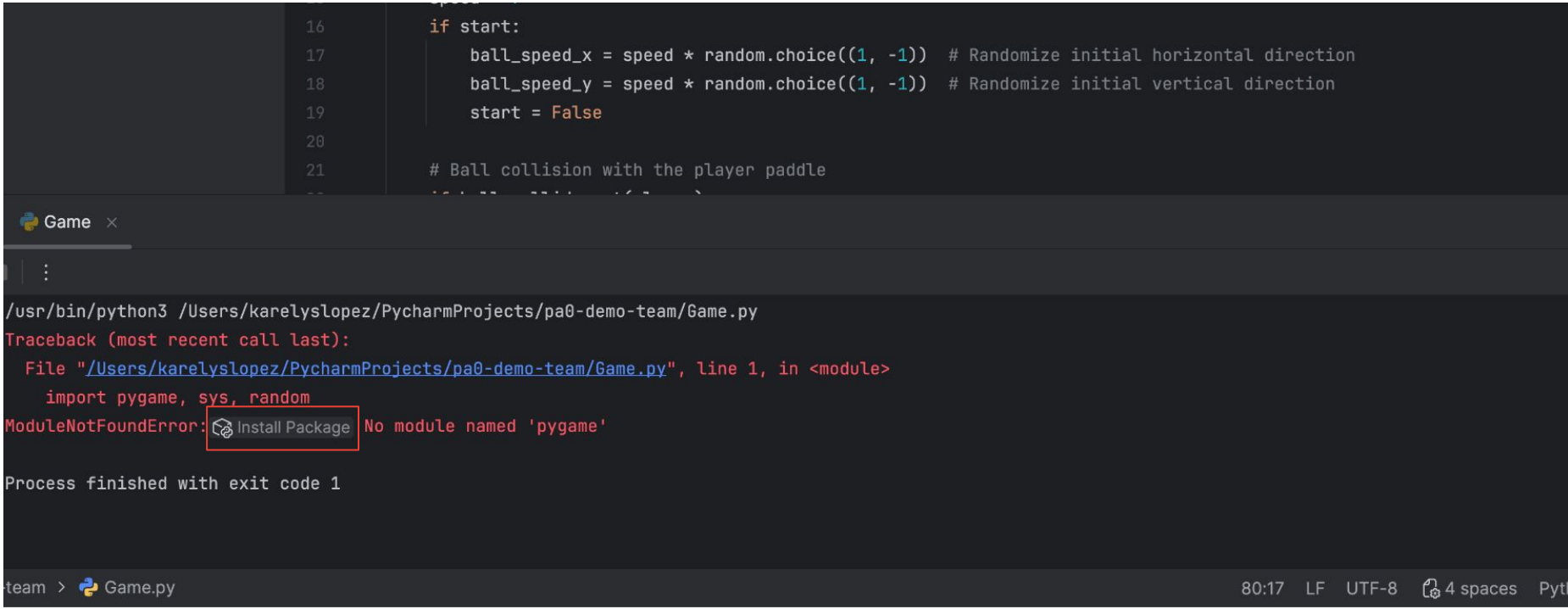
# If you selected commit only and want to push



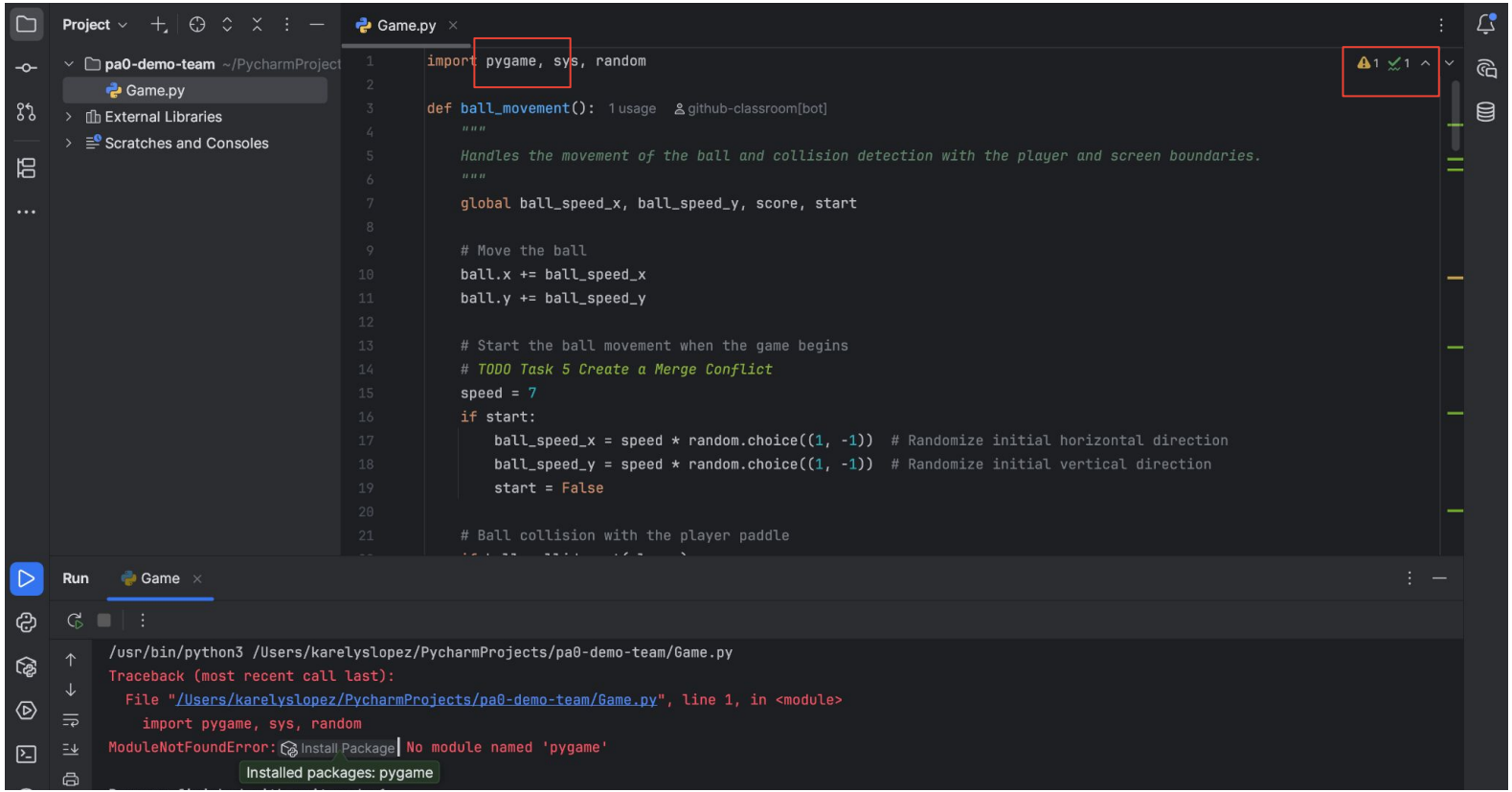
# Running the project for the first time



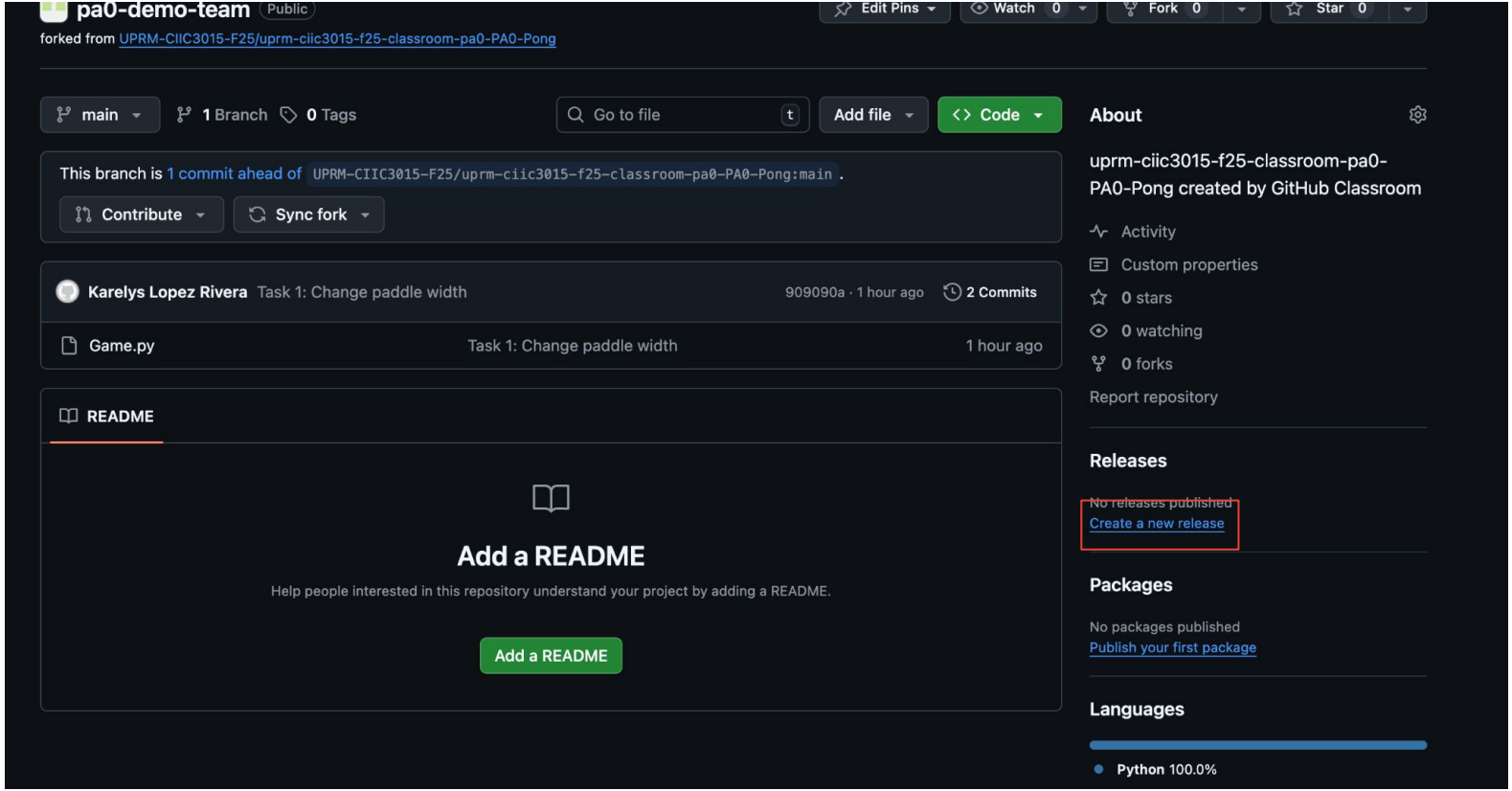
# Running the project for the first time


16 if start:
17 ball_speed_x = speed * random.choice((1, -1)) # Randomize initial horizontal direction
18 ball_speed_y = speed * random.choice((1, -1)) # Randomize initial vertical direction
19 start = False
20
21 # Ball collision with the player paddle
Game
/usr/bin/python3 /Users/karelyslopez/PycharmProjects/pa0-demo-team/Game.py
Traceback (most recent call last):
File "/Users/karelyslopez/PycharmProjects/pa0-demo-team/Game.py", line 1, in <module>
import pygame, sys, random
ModuleNotFoundError: Install Package No module named 'pygame'
Process finished with exit code 1
team > Game.py
80:17 LF UTF-8 4 spaces

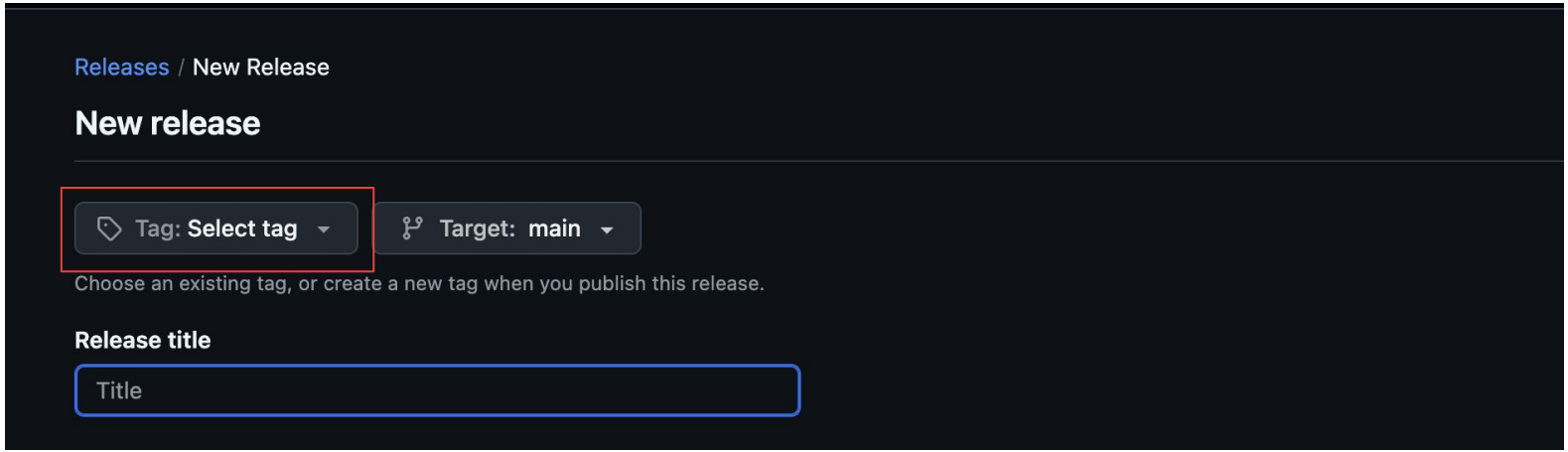
# Running the project for the first time


Project
pa0-demo-team ~/PycharmProject
Game.py
External Libraries
Scratches and Consoles
Game.py
import pygame, sys, random
def ball_movement(): 1 usage github-classroom[bot]
"""
Handles the movement of the ball and collision detection with the player and screen boundaries.
"""
global ball_speed_x, ball_speed_y, score, start
# Move the ball
ball.x += ball_speed_x
ball.y += ball_speed_y
# Start the ball movement when the game begins
# TODO Task 5 Create a Merge Conflict
speed = 7
if start:
ball_speed_x = speed * random.choice((1, -1)) # Randomize initial horizontal direction
ball_speed_y = speed * random.choice((1, -1)) # Randomize initial vertical direction
start = False
# Ball collision with the player paddle
Run
Game
/usr/bin/python3 /Users/karelyslopez/PycharmProjects/pa0-demo-team/Game.py
Traceback (most recent call last):
File "/Users/karelyslopez/PycharmProjects/pa0-demo-team/Game.py", line 1, in <module>
import pygame, sys, random
ModuleNotFoundError: Install Package No module named 'pygame'
Installed packages: pygame

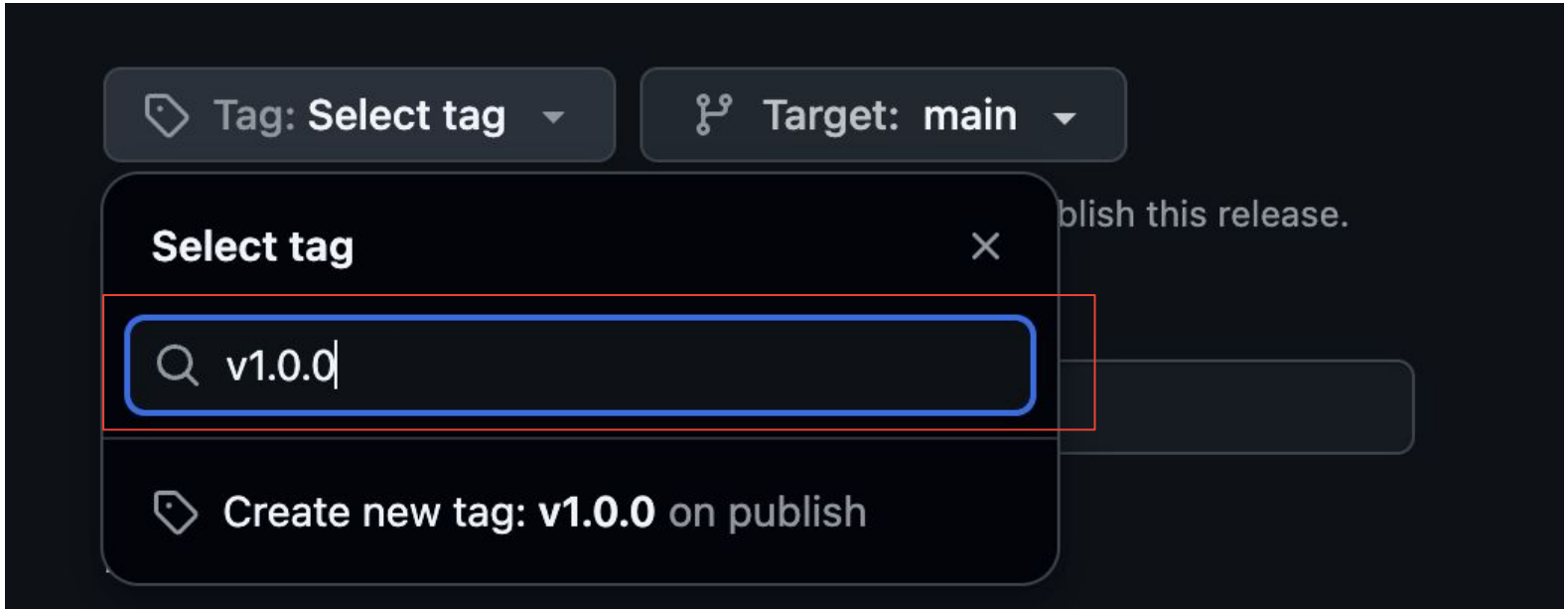
# Creating a Release



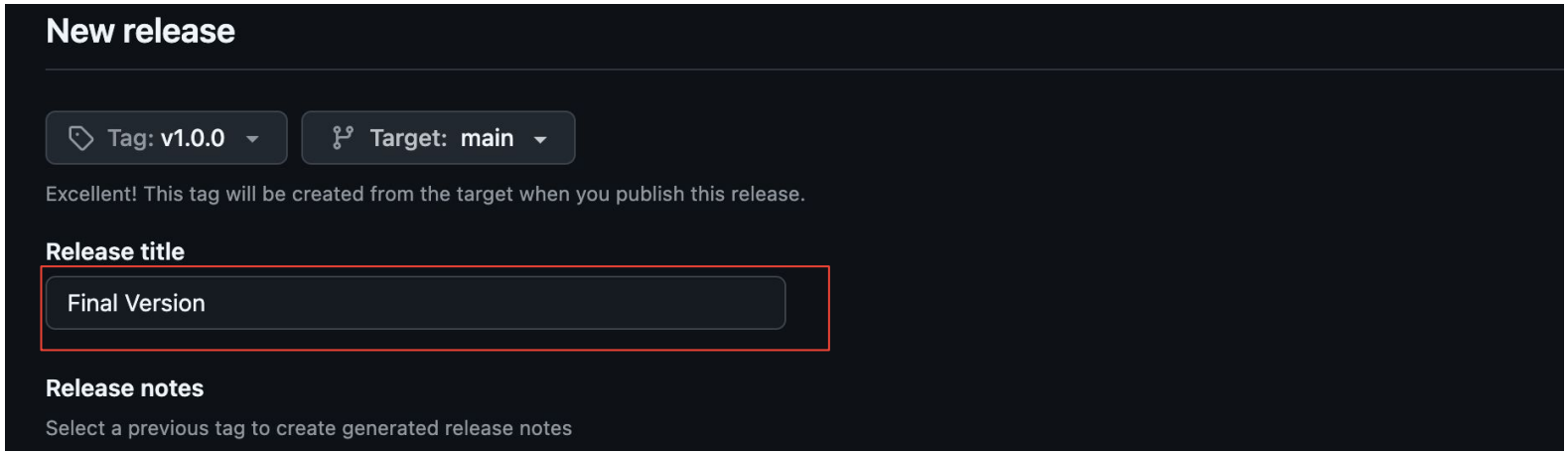
# Creating a Release (One per team)


Releases / New Release
New release
Tag: Select tag
Target: main
Choose an existing tag, or create a new tag when you publish this release.
Release title
Title

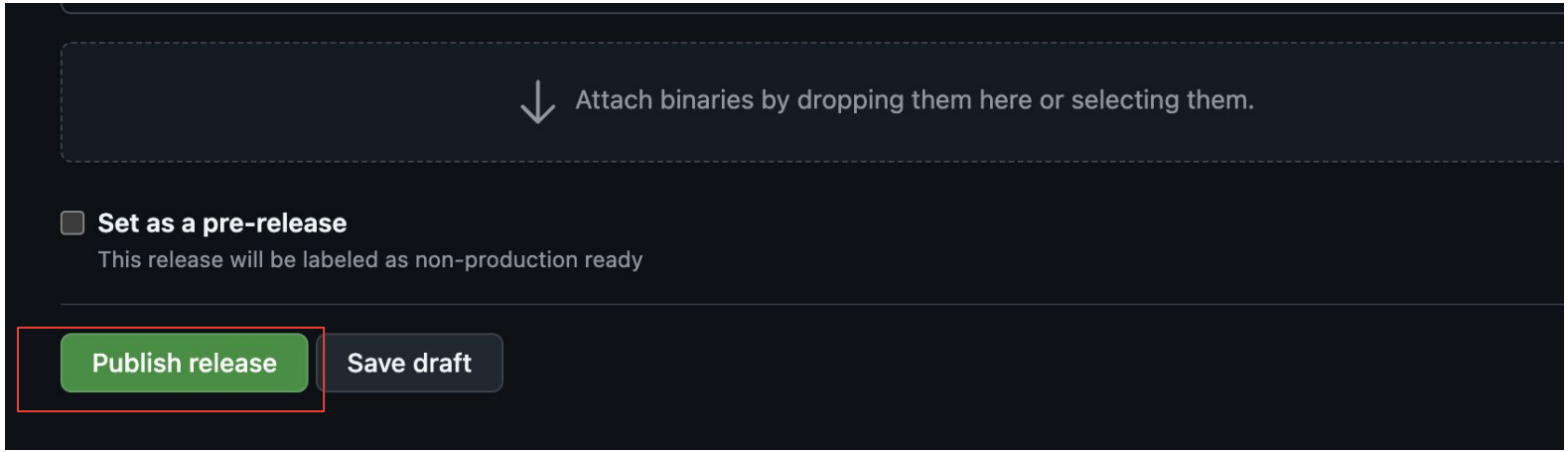
# Creating a Release



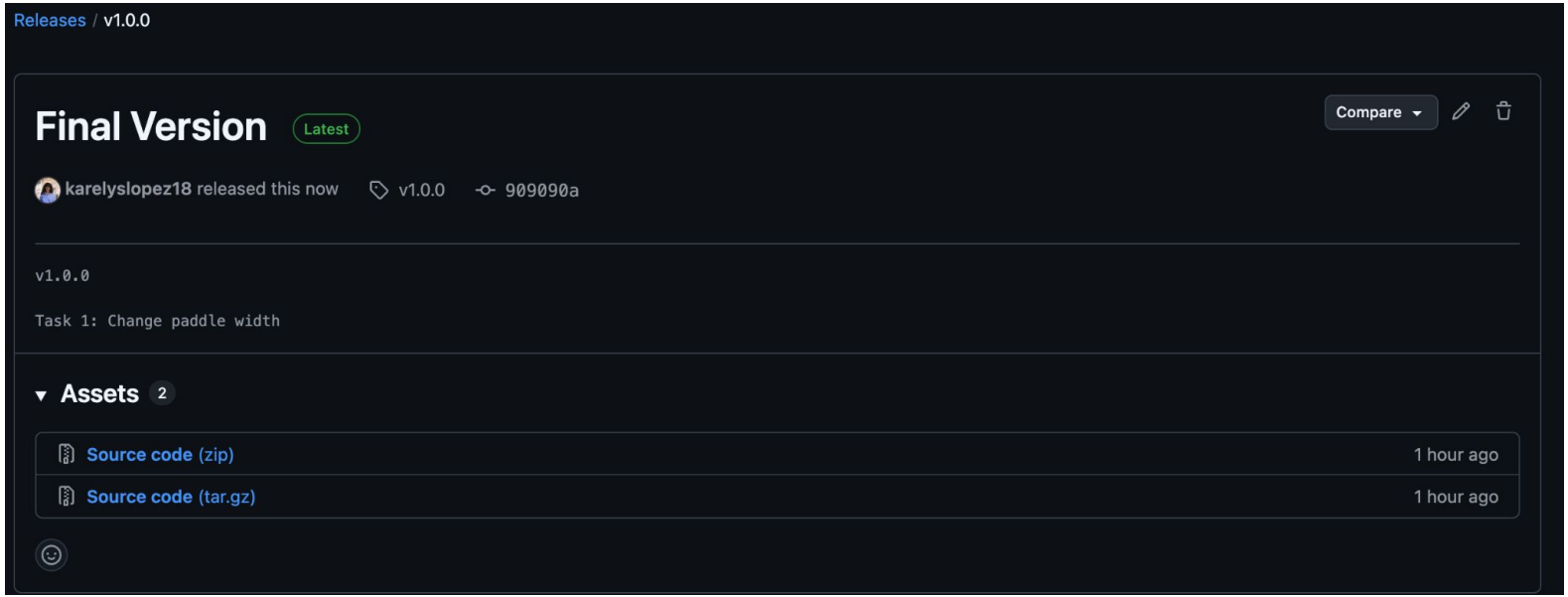
# Creating a Release


New release
Tag: v1.0.0
Target: main
Excellent! This tag will be created from the target when you publish this release.
Release title
Final Version
Release notes
Select a previous tag to create generated release notes

# Creating a Release


Attach binaries by dropping them here or selecting them.
Set as a pre-release
This release will be labeled as non-production ready
Publish release
Save draft

# Creating a Release



CODE PATH *ORG

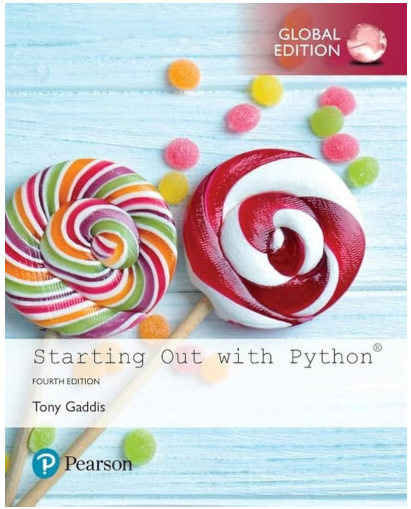
# Loops

## CIIC 3015

Karelys López Rivera

# Lecture Goals

- **Loops**
  - What are they?
  - Why do we need them?
- **While Loops**
  - How to create and use them?
  - Demo
- **Sequences**
  - What are they?
- **For Loops**
  - How to create and use them?
  - Demo




**Starting Out With Python (4th Edition) Chapter 4**

# Recap

# Functions

A **function** is a reusable block of code that performs a specific task such as manipulating data, making calculations, or processing text.

The benefits of using functions include:

- Reusability: Functions allow you to write a block of code once and use it multiple times, saving time and effort.
- Readability: Functions break down complex code into smaller, more manageable chunks, making your code easier to understand and maintain.
- Reduced errors: Using functions helps you avoid introducing errors by duplicating code.

# Parameters and arguments

A **parameter** is a variable defined in the header of a function definition.

- Acts as a placeholder for a value that will be passed into the function when it is called
- Listed inside the parentheses following the function name

An **argument** is a value supplied to a function when it is called.

- Passed into the function's parameters
- Listed inside the parentheses when you call the function

Parameter

```
def greet(name):
    print(f"Hello, {name}!")
```

Argument

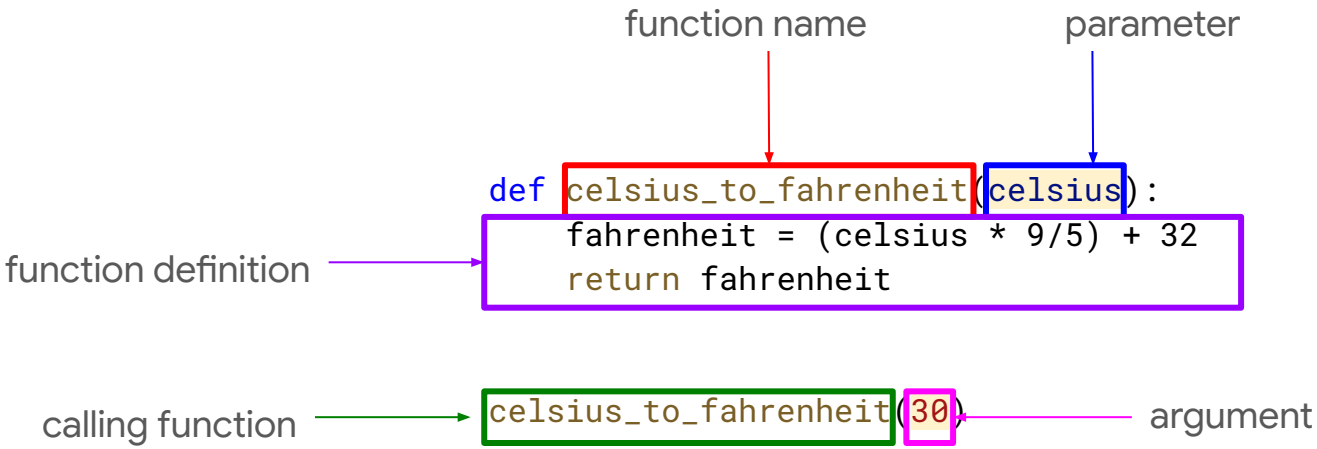
```
greet("Rosario")
```

# Return statements

A **return statement** is a line of code that returns a value back from a function to the location in the program where the function was called.

- Functions can return a value using the `return` keyword.
- This value can be stored in a variable and used later in your program
- The syntax of a `return` statement is the keyword `return` followed by the value that you want to return from the function.

# Example:


function name
parameter
def celsius_to_fahrenheit(celsius):
fahrenheit = (celsius * 9/5) + 32
return fahrenheit
function definition
calling function
celsius_to_fahrenheit(30)
argument

# Example:

```
# In test_module.py
def convert_celsius_to_fahrenheit(celsius):
    fahrenheit = 9 / 5 * celsius + 32
    return fahrenheit
```

```
# In file_that_uses_test_module.py
import test_module
print(test_module.convert_celsius_to_fahrenheit(40))
```

# Scenario

# Loops: Motivating Example

- So far in the class, our code has had to run just once and then we have to manually press the “Run” button if we want our code to run again.
- However, most real world applications don’t just “Run” once; they keep going indefinitely until we tell it to stop or we close the program ourselves (like closing an app in our phones).

# Loops: Motivating Example

- An example of this is a generic login page. We have to type in the username and password.
- But what happens if the password is incorrect?

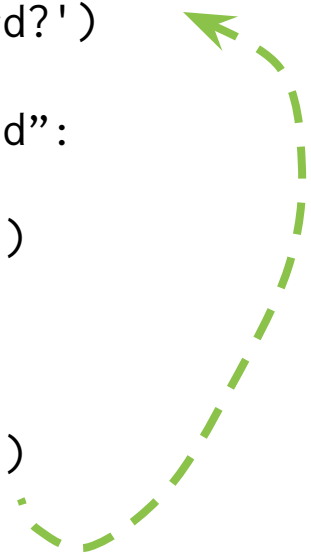
```
password = input('Password?')

if password == “helloworld”:

  print('Access granted!')

else:

  print('Wrong password.')
```

You only get one try!

By: Jean Carlos Mendez

# Loops: Motivating Example

```
password = input('Password?')

if password == “helloworld”:

  print('Access granted!')

else:

  print('Wrong password.')
```



*Should take new input and check it until the user gets it right*

# Loops: Motivating Example

```
password = input('Password?')

if password == 'helloworld':

  print('Access granted!')

else:

  print('Wrong password.')
```

# Loops: Motivating Example

```
password = ''
while password != 'helloworld':
    password = input('Password?')
    if password == 'helloworld':
      print('Access granted!')
    else:
      print('Wrong password.')
```

# Loops: Motivating Example

```
password = ''
while password != 'helloworld':
    password = input('Password?')
    if password == 'helloworld':
      print('Access granted!')
    else:
      print('Wrong password.')
```

while the password isn't correct

# Loops: Motivating Example

```
password = ''
while password != 'helloworld':
    password = input('Password?')
    if password == 'helloworld':
      print('Access granted!')
    else:
      print('Wrong password.')
```

keep asking for the password

# Loops: Motivating Example

```
password = ''
while password != 'helloworld':
    password = input('Password?')
    if password == 'helloworld':
      print('Access granted!')
    else:
      print('Wrong password.')
```

need an initial value

# If vs. while

**If** the lights are on, shout "LIGHTS!".

**While** the lights are on, shout "LIGHTS!".

**If** you've clapped your hands less than 5 times, clap your hands.

**While** you've clapped your hands less than 5 times, clap your hands.

**If** you like tacos, give me a thumbs up.

**While** you like tacos, give me a thumbs up.

By: Jean Carlos Mendez

Google | GR |

# while loops

# Introduction: `while` loop

A **`while` loop** repeatedly executes a set of instructions.

- Iterates until the loop condition evaluates to `False`
- Used when the number of iterations is unknown

```
num = 1

while num <= 5:
    print(num)
    num += 1
```

```
1
2
3
4
5
```

# Syntax: `while` loop

The general syntax of a `while` loop:

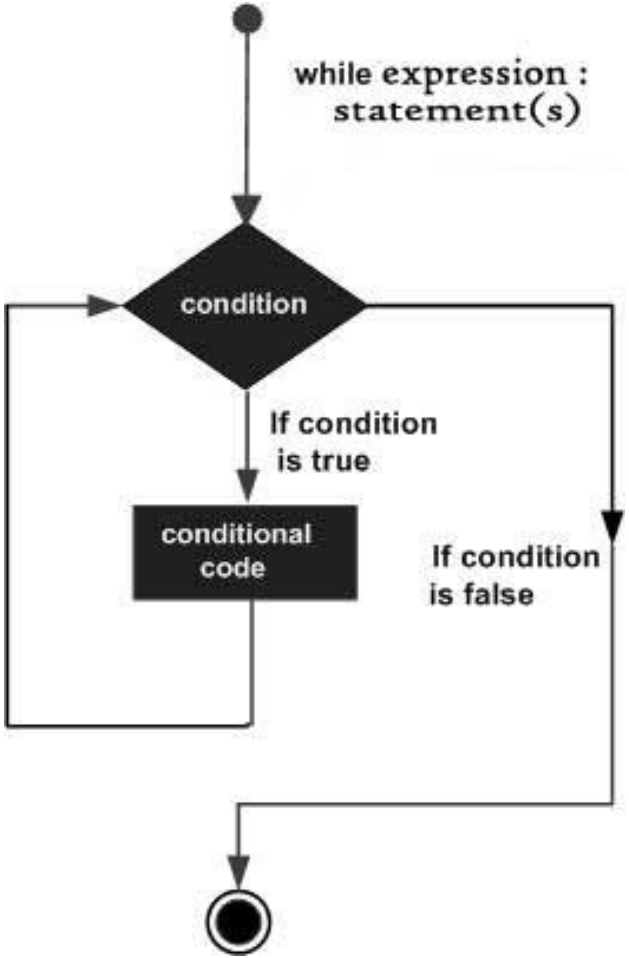
- Start a `while` loop by placing the keyword `while` in the header
- After the keyword `while`, place the expression that will be evaluated on each iteration
- Place a colon (`:`) at the end of the header
- Indent the body of the loop

```
number = 0

while number != -1:
    number = int(input("Enter a number (or -1 to quit): "))
        print(f"You entered: {number}")
```

# while loops

Control structure that runs a block of code *repeatedly* until a certain condition is no longer met

```
while condition is true:

  do something
```




By: Jean Carlos Mendez

# Examine the code

What is the output of the following code block?

```
num = 2

while num <= 10:
    print(num)
    num += 2
```

# Examine the code

What is the output of the following code block?

```
num = 2

while num <= 10:
    print(num)
    num += 2
```

```
2
4
6
8
10
```

# Check your understanding

What is the output of this code block?

```
views = 1

while views <= 3:
    print(f"Replays left: {3 - views}")
    views += 1
```

# Check your understanding

What is the output of this code block?

```
views = 1

while views <= 3:
    print(f"Replays left: {3 - views}")
    views += 1
```

```
Replays left: 2
Replays left: 1
Replays left: 0
```

# IT'S CODING TIME

- Print numbers 1 through 100
- Verify Password
- Countdown

# Questions?

Post to our Discord server

or

email:

upr-ciic3015-staff@upr.edu

karelys.lopez1@upr.edu

Give us anonymous feedback!

https://bit.ly/CIIC3015_F25_Feedback

CODE PATH .ORG

# Recap

# Introduction: `while` loop

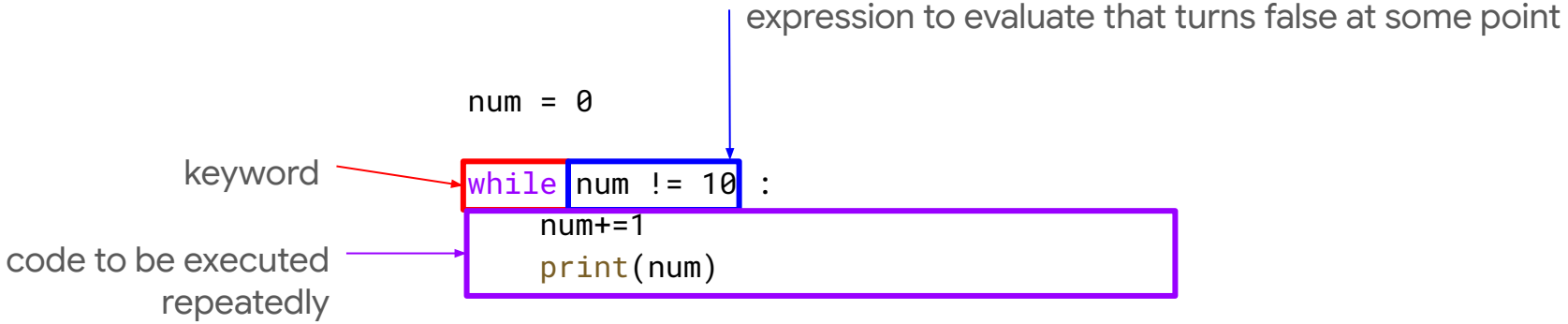
A **`while` loop** repeatedly executes a set of instructions.

- Iterates until the loop condition evaluates to `False`
- Used when the number of iterations is unknown

```
num = 1

while num <= 5:
    print(num)
    num += 1
```

```
1
2
3
4
5
```

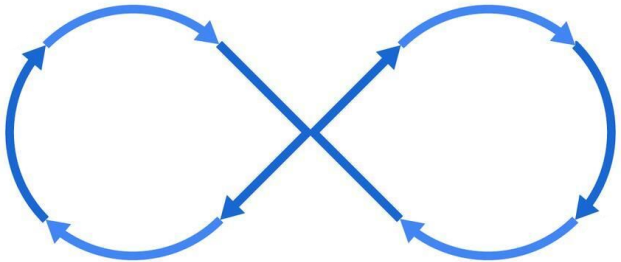
# Syntax: `while` loop



# infinite loops

# Infinite loops

- A `while` loop that never terminates is called an **infinite loop**
- To avoid infinite loops:
  - Make sure the loop expression can eventually evaluate to `False`
  - And/or utilize a `break` statement



# Example: Infinite loop

```
num = 10

while num > 0:
    print(num)
print("Blast off!")
```

```
10
10
10
10
10
(Infinite loop: will not stop printing 10. . . )
```

# break and continue in while loops

# break

**break** is a keyword used to terminate a loop early if needed, whenever the code hits a break it will exit the loop immediately.

For example:

```
while True:
  word = input("Type 'stop' to exit: ")
  if word == "stop":
    break
  print("Continuing")

print("Finished loop!")
```

# while loops: break keyword

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

By: Jean Carlos Mendez

# while loops: break keyword

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit:
```

By: Jean Carlos Mendez

# while loops: break keyword

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

By: Jean Carlos Mendez

# while loops: break keyword

word == "Hello"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "Hello"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "Hello"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
```

By: Jean Carlos Mendez

# while loops: break keyword

`word == "Hello"`

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "Hello"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "Hello"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "Hello"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
Type 'stop' to exit:
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "stop"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
Type 'stop' to exit: stop
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "stop"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
Type 'stop' to exit: stop
```

By: Jean Carlos Mendez

# while loops: break keyword

`word == "stop"`

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
Type 'stop' to exit: stop
```

By: Jean Carlos Mendez

# while loops: break keyword

word == "stop"

We can break out of loops early if we need to, using the break keyword

```
while True:
    word = input("Type 'stop' to exit: ")
    if word == "stop":
        break
    print("Continuing")

print("Finished loop!")
```

Whenever we hit a break, we immediately exit the loop

```
Type 'stop' to exit: Hello
Continuing
Type 'stop' to exit: stop
Finished loop!
```

By: Jean Carlos Mendez

# IT'S CODING TIME

- Print numbers 1 until reached number 20
- Verify Password with a max attempt of 3

# Questions?

Post to our Discord server

or

email:

upr-ciic3015-staff@upr.edu

karelys.lopez1@upr.edu

Give us anonymous feedback!

https://bit.ly/CIIC3015_F25_Feedback

CODE PATH .ORG

# Sequences

# Sequences

So far in the class we have seen “strings”. A **string** is a sequence of characters used for representing textual data.

- Strings can contain characters such as letters, numbers, symbols, and whitespaces.
- A string that does not contain any characters is an **empty string** (`""`).

**String examples**

```
"Hello world!"
"wall"
"88 + 99"
"&%#@"
""
```

# Strings

- A **string** is a sequence of characters.

```
'wall'  '88 + 99'  'Click here.'  '   '
```

- A **character** is anything typed in a single keystroke, commonly known as a **char**.

```
'r'  '8'  'A'  ','  '&'  ' '  '?'
```

By: Jean Carlos Mendez

Google | GDR |

# Strings: Length

- What is the length of each string?

| | |
|---|---|
| 'UPRM' | 4 |
| ':)' | 2 |
| 'Good morning!' | 13 |
| '$10.99' | 6 |
| '2 + 2 = 5' | 9 |

By: Jean Carlos Mendez

# The `len()` function

The **`len()`** function returns the number of characters in a string.

```
print(len("UPRM"))
4

print(len(":)"))
2

print(len("Good Morning!"))
13
```

```
print(len("$10.99"))
6

print(len("2 + 2 = 5"))
9

print(len(""))
0                    # Length of empty string = 0
```

By: Jean Carlos Mendez

# Indexing

An **index** is a number that represents the position of an element in a sequence, such as a string or a list.

- In a string, an index indicates the position of each character in the string.
- Indexing starts at 0. The first character in a string has an index of 0, the second an index of 1, and so on.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| H | e | l | l | o | w | o | r | l | d |

# Strings: Index

| | |
|---|---|
| 'Chipotle' | C: 0, p: 3 |
| 'Harry Potter' | P: 6, r: 11 |
| 'Nashville, TN' | a: 1, ,: 9 |
| '$4593.01' | 5: 2, 0: 6 |
| '...what?' | w: 3, ?: 7 |

By: Jean Carlos Mendez

Google | GTR |

# Access characters by index

Place square brackets ([ ]) after the string variable to access characters by index.

```
phrase = "Happy Birthday!"

print(phrase[0])            # Index 0: first character
print(phrase[1])            # Index 1: second character
print(phrase[5])            # Index 5: fourth character
print(phrase[14])           # Index 14: last character
```

# IndexError

When we try to access an index value greater than the length-1 of a string we get an error

```
phrase = "cat"

print(phrase[6])
```

```
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
IndexError: string index out of range
```

# Negative indexing

Use **negative indexing** to access characters from the end of the string.

- The last character in the string has an index of -1, the second to last character an index of -2, etc.

```
phrase = "Happy Birthday!"

print(phrase[-1])          # Index -1: last character
print(phrase[-2])          # Index -2: second to last character
print(phrase[-10])         # Index -10: tenth to last character
print(phrase[-15])         # Index -15: first character
print(phrase[-16])         # Index -16: out of range
```

```
!
y

H
IndexError: string index out of range
```

# in operator

in operator: checks if value exists in sequence

in expression evaluates to True if value exists

```
word = 'banana'
if 'n' in word:
  print('Found an n')
else:
  print('Did not find a n')
```



Google | GDSC | UTEC

# in operator: Practice

```
'N' in 'Nashville, TN'    True
'n' in 'Nashville, TN'    False
char = ' '
char in 'Intro to Computer Science'  True
word = 'CS104'
char in word  False
```

By: Jean Carlos Mendez

# IT'S CODING TIME

- Get length of a string using len()
- Indexing
- in operator

# Questions?

Post to our Discord server

or

email:

upr-ciic3015-staff@upr.edu

karelys.lopez1@upr.edu

Give us anonymous feedback!

https://bit.ly/CIIC3015_F25_Feedback

CODE PATH .ORG

# Recap

# Sequences

So far in the class we have seen “strings”. A **string** is a sequence of characters used for representing textual data.

- Strings can contain characters such as letters, numbers, symbols, and whitespaces.
- A string that does not contain any characters is an **empty string** (`""`).

**String examples**

```
"Hello world!"
"wall"
"88 + 99"
"&%#@"
""
```

# The `len()` function

The **`len()`** function returns the number of characters in a string.

```
print(len("UPRM"))
4

print(len(":)"))
2

print(len("Good Morning!"))
13
```

```
print(len("$10.99"))
6

print(len("2 + 2 = 5"))
9

print(len(""))
0                    # Length of empty string = 0
```

By: Jean Carlos Mendez

# Access characters by index

Place square brackets ([ ]) after the string variable to access characters by index.

```
phrase = "Happy Birthday!"

print(phrase[0])            # Index 0: first character
print(phrase[1])            # Index 1: second character
print(phrase[5])            # Index 5: fourth character
print(phrase[14])           # Index 14: last character
```

# in operator

in operator: checks if value exists in sequence

in expression evaluates to True if value exists

```
word = 'banana'
if 'n' in word:
  print('Found an n')
else:
  print('Did not find a n')
```



Google | GDSC | UTEC

scenario

# for loops: Motivation

- Let's say that we have the string "Colegio" and we want to print each letter individually.
- How could we do this with a **while** loop?

```
name = 'Colegio'

index = 0

while index < len(name):

    print(name[index])

    index = index + 1
```

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

Variables

Output

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

Variables

name = ‘Colegio’

Output

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

0 < 7? True!
Step into loop

Variables

name = ‘Colegio’

index = 0

Output

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

Variables

```
name = ‘Colegio’
index = 0
```

Output

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

**Variables**

```
name = ‘Colegio’
index = 0
```

**Output**

```
> C
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

1 < 7? True!
Step into loop

Variables

```
name = ‘Colegio’
index = 1
```

Output

```
> C
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```



Variables

```
name = ‘Colegio’
index = 1
```

Output

```
> C
```

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

**Variables**

```
name = ‘Colegio’
index = 1
```

**Output**

```
> C
> o
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

2 < 7? True!
Step into loop

Variables

```
name = ‘Colegio’
index = 2
```

Output

```
> C
> o
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```



Variables

```
name = ‘Colegio’
index = 2
```

Output

```
> C
> o
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

Variables

```
name = ‘Colegio’
index = 2
```

Output

```
> C
> o
> l
```

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

3 < 7? True!
Step into loop

Variables

```
name = ‘Colegio’
index = 3
```

Output

```
> C
> o
> l
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```



Variables

```
name = ‘Colegio’
index = 3
```

Output

```
> C
> o
> l
```

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

**Variables**

```
name = ‘Colegio’
index = 3
```

**Output**

```
> C
> o
> l
> e
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

4 < 7? True!
Step into loop

Variables

```
name = ‘Colegio’
index = 4
```

Output

```
> C
> o
> l
> e
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

**Variables**

```
name = ‘Colegio’
index = 4
```

**Output**

```
> C
> o
> l
> e
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

**Variables**

```
name = ‘Colegio’
index = 4
```

**Output**

```
> C
> o
> l
> e
> g
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

5 < 7? True!
Step into loop

**Variables**

```
name = ‘Colegio’
index = 5
```

**Output**

```
> C
> o
> l
> e
> g
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```



Variables

```
name = ‘Colegio’
index = 5
```

Output

```
> C
> o
> l
> e
> g
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

**Variables**

```
name = ‘Colegio’
index = 5
```

**Output**

```
> C
> o
> l
> e
> g
> i
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

6 < 7? True!
Step into loop

**Variables**

```
name = ‘Colegio’
index = 6
```

**Output**

```
> C
> o
> l
> e
> g
> i
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```



Variables

```
name = ‘Colegio’
index = 6
```

Output

```
> C
> o
> l
> e
> g
> i
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

**Variables**

```
name = ‘Colegio’
index = 6
```

**Output**

```
> C
> o
> l
> e
> g
> i
> o
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

7 < 7? False!
Skip the Loop!

**Variables**

```
name = ‘Colegio’
index = 7
```

**Output**

```
> C
> o
> l
> e
> g
> i
> o
```

By: Jean Carlos Mendez

# for loops: Motivation

```
name = 'Colegio'
index = 0
while index < len(name):
    print(name[index])
    index = index + 1
```

Variables

```
name = ‘Colegio’
index = 7
```

Output

```
> C
> o
> l
> e
> g
> i
> o
```

# for loops: Motivation

- This pattern of iterating through every element of a sequence is ***EXTREMELY COMMON***.
- It's so common, in fact, that another loop was created to simplify these iterations.
- This loop is called the **for** loop, with the name coming from "**for** every element in this sequence, do ..."

# for loops

# Introduction: `for` loop

A **`for` loop** is used for iterating over a sequence.

- Iteration is done in order
- Used when the number of iterations is known

**One type of sequence is a string:**

```
for char in "Python":
    print(char)
```

```
P
y
t
h
o
n
```

# Syntax: `for` loop

The general syntax of a `for` loop:

- Start a `for` loop by placing the keyword `for` in the header
- Place a colon (`:`) at the end of the header
- Indent the body of the loop

```
for char in "Python":
    print(char)
```

# while vs. for loops

- Similarity: repeating code in a loop
- Difference:
  - while loop goes until condition is no longer met
  - for loop goes for a predetermined / known number of times



Google | GR |

# The sequence in a `for` loop

Place the sequence to iterate over after the keyword `in`

This `for` loop will iterate through the sequence of characters of the string

```
for letter in "gold":
    print(letter)
```

# The loop variable

Place the loop variable after the keyword `for`

The variable stores the data from the sequence during each iteration.

Use the variable in the body of the loop to perform actions

```
for letter in "gold":
    print(letter)
```

# Examine the code

What is the output of each code block?

```
for char in "Code":
    print(char)
```

# Examine the code

What is the output of each code block?

```
for char in "Code":
    print(char)
```

```
C
o
d
e
```

# the range() function

# The `range()` function

**`range()`** is a built-in Python function that generates a sequence of integers.

- One way to include a sequence of integers in a `for` loop is to use `range()`:

```
for i in range(5, 56, 10)
    print(i)
```

```
5
10
15
25
35
45
55
```

# Syntax: `range()`

`range()` can take up to three arguments: a `start` value, `stop` value, and `step` value.

- `start` value (optional): specifies the starting value (inclusive)
  - In `range(5, 56, 10)`, the sequence starts at `5`
- `stop` value (required): specifies the ending value (exclusive)
  - In `range(5, 56, 10)`, the sequence ends at `55` because `56` is excluded
- `step` value (optional): specifies the increment or decrement between values
  - In `range(5, 56, 10)`, the sequence increments by `10`

# `range(stop)`

In the example:

- Required `stop` value is `56`
- Generates numbers between `0` and `55`
- The `stop` value is excluded from the sequence.

```
range(56)
```

# `range(start, stop)`

In this example:

- Required `stop` value is 56
- Optional `start` value is 5
- Generates numbers between 5 and 55
- The `stop` value is excluded from the sequence, but the `start` value is included.

```
range(5, 56)
```

# `range(start, stop, step)`

In this example;

- Required `stop` value is `56`
- Optional `start` value is `5`
- Optional `step` value is `10`
- Generates numbers between 5 and 55 in increments of `10`
- The `stop` value is excluded from the sequence, but the `start` value is included.

```
range(5, 56, 10)
```

# Examine the code

What is the output of this code block?

```
for i in range(4, 20, 5):
    print(i)
```

# Examine the code

What is the output of this code block?

```
for i in range(4, 20, 5):
    print(i)
```

```
4
9
14
19
```

# Negative `step` value

In this example:

- Required `stop` value is `89`
- Optional `start` value is `100`
- Optional `step` value is `-2`
- Generates numbers between `100` and `90` in decrements of 2
- The `stop` value is excluded from the sequence, but the `start` value is included.

```
range(100, 89, -2)
```

# Check your understanding

What is the output of this code block?

```
for i in range(30, 5, -5):
    print(i)
```

# Check your understanding

What is the output of this code block?

```
for i in range(30, 5, -5):
    print(i)
```

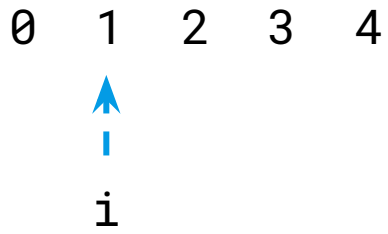
```
30
25
20
15
10
```

# for loops: range() function

```
for i in range(5):
    print(i)
```

Google

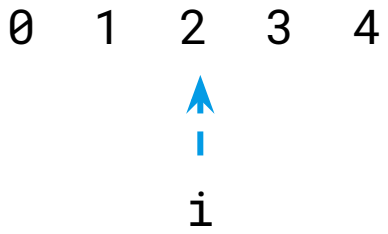
# for loops: range() function




```
for i in range(5):
    print(i)
```

```
> 0
> 1
```

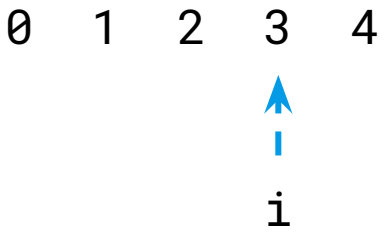
# for loops: range() function




```
for i in range(5):
    print(i)
```

```
> 0
> 1
> 2
```

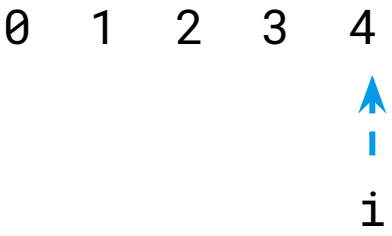
# for loops: range() function




```
for i in range(5):
    print(i)
```

```
> 0
> 1
> 2
> 3
```

# for loops: range() function



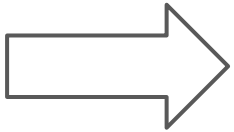

```
for i in range(5):
    print(i)
```

```
> 0
> 1
> 2
> 3
> 4
```

Google

# for loops: range() function

```
print("Hello world")
print("Hello world")
print("Hello world")
```



(range(3) == [0,1,2])

```
for i in range(3):
    print("Hello world")
```

Notice: you don’t have to use the variable if you don’t need it! You can use the range to control how many times the body is executed.

# IT'S CODING TIME

- homemade in()
- Print all the numbers from 0 to N
- multiplication table
- count the vowels

# Questions?

Post to our Discord server

or

email:

upr-ciic3015-staff@upr.edu

karelys.lopez1@upr.edu

Give us anonymous feedback!

https://bit.ly/CIIC3015_F25_Feedback

CODE PATH .ORG